

AD-A078 937

INTERNATIONAL COMPUTING CO BETHESDA MD  
VESSEL TRAFFIC SERVICES PROCESSING/DISPLAY SUBSYSTEM SOFTWARE R--ETC(U)  
SEP 79 C C HENSON , R S GRAHAM , B A MCINTOSH DOT-CG-81-78-1833

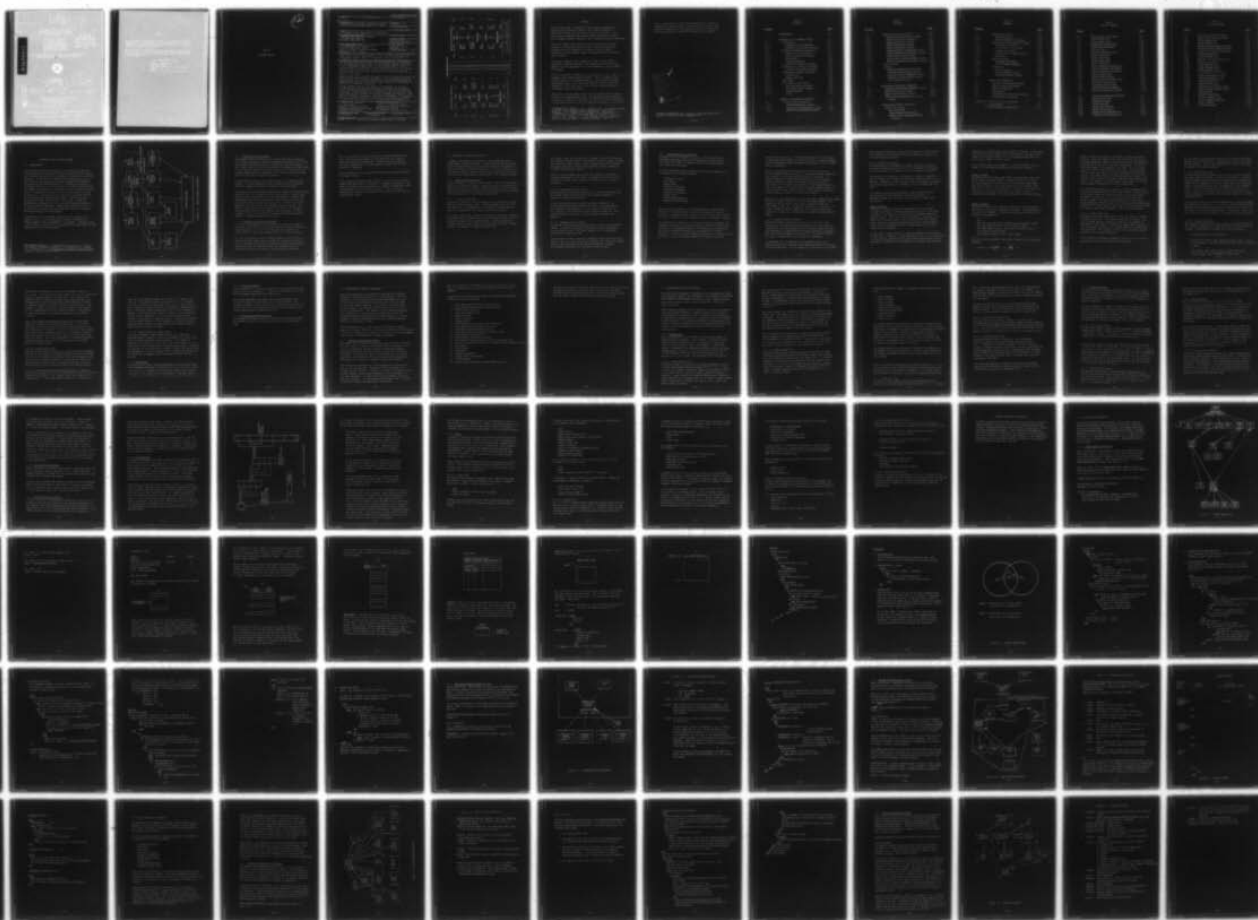
F/G 15/5

USCG -D-73-79

NL

UNCLASSIFIED

1 OF 5  
AD-  
A078937





REPORT NO. CG-D-73-79

LEVEL

ADM 052  
R076501

12

VESSEL TRAFFIC SERVICES  
PROCESSING/DISPLAY SUBSYSTEM  
SOFTWARE REQUIREMENTS AND DESIGN

Part 2. Software Design

10 CARLE C. HENSON  
R. SCOTT GRAHAM  
BONNIE A. MCINTOSH  
International Computing  
4330 East-West Highway  
Bethesda, Maryland 20014

DDC  
RECEIVED  
DEC 18 1979  
E

U.S. Coast Guard Research and Development Center  
Avery Point Groton Connecticut 06340



11 SEPTEMBER 1979

12 426  
9 FINAL REPORT, Dec 78 - Sep 79

18 USCG

19 D-73-79

12 17 056

Document is available to the U.S. public through  
the National Technical Information Service  
Springfield, Virginia 221

DDC FILE COPY

15 DOT-CG-81-78-1883

PREPARED FOR

U.S. DEPARTMENT OF TRANSPORTATION  
UNITED STATES COAST GUARD

OFFICE OF RESEARCH AND DEVELOPMENT

WASHINGTON D. C. 20590

393469

AD A078937

A061307

NOTICE

This document is disseminated under the sponsorship of the Department of Transportation in the interest of information exchange. The United States Government assumes no liability for its contents or use thereof.

The United States Government does not endorse products or manufacturers. Trade or manufacturers' names appear herein solely because they are considered essential to the object of this report.


*D.L. Birkimer*

DONALD L. BIRKIMER, Ph.D., P.E.  
Technical Director  
U.S. Coast Guard Research and Development  
Center  
Avery Point, Groton, Connecticut 06340

12

PART II  
SOFTWARE DESIGN



1. Report No. 24 BCG-D-73-79	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle VESSEL TRAFFIC SERVICES PROCESSING/DISPLAY SUBSYSTEM SOFTWARE REQUIREMENTS AND DESIGN	5. Report Date September 1979	6. Performing Organization Code
7. Author(s) C.C. Henson, R.S. Graham and B.A. McIntosh	8. Performing Organization Report No.	
9. Performing Organization Name and Address International Computing Company 4330 East-West Highway Bethesda, Maryland 20014	10. Work Unit No. (TRAIS)	11. Contract or Grant No. DOT-CG-81-78-1833
12. Sponsoring Agency Name and Address U.S. Department of Transportation United States Coast Guard Office of Research and Development Washington, D.C. 20590	13. Type of Report and Period Covered Final Report December 1978 to September 1979	14. Sponsoring Agency Code
15. Supplementary Notes The contract under which this report was submitted was under the technical supervision of the Coast Guard Research and Development Center, Groton, Connecticut, 06340. R&DC 24/79		
16. Abstract This report defines the detailed operational requirements for the Vessel Traffic System (VTS) Processing/Display Subsystem and provides the current state of the software design. Both parts (i.e., requirements and design) of this report begin with an overview and an introduction to the methodology used. The balance of both parts is a detailed, technically oriented, function by function description of the specification on the one hand and the design on the other.  Specifications are included for operator invoked functions which are used to enter and retrieve information, and to request subsystem services; as well as specifications for automatic processes which update vessel location, course and speed, and detect and report on potentially hazardous conditions.  At the highest level of the design, the software for the VTS Processing/Display Subsystem has been decomposed into processes which operate concurrently and cooperate to perform the required functions. These processes will communicate and synchronize their activities by the exchange of explicit messages and answers. This approach provides flexibility in assigning processes to processors, which is a critical factor in providing a system that can manage a broad range of functional and load handling requirements. 		
17. Key Words State Machines VTS Data Bases Operator Invoked Functions Automatic Background Processes Simulation and Testing	18. Distribution Statement Document is available to the U.S. public through the National Technical Information Service, Springfield, Virginia 22161	
19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of Pages 771
22. Price		

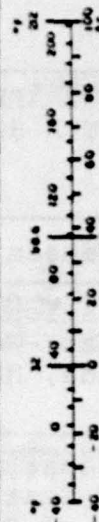
# METRIC CONVERSION FACTORS

## Approximate Conversions from Metric Measures

Symbol	When You Know	Multiply by	To Find	Symbol
<b>LENGTH</b>				
m	meters	39.37	inches	in
cm	centimeters	2.54	inches	in
mm	millimeters	0.03937	inches	in
m	meters	1.0936	yards	yd
km	kilometers	0.6214	miles	mi
<b>AREA</b>				
m <sup>2</sup>	square meters	1.196	square yards	sq yd
cm <sup>2</sup>	square centimeters	1.55	square inches	sq in
ha	hectares	2.471	acres	ac
<b>MASS (weight)</b>				
g	grams	0.03523	ounces	oz
kg	kilograms	2.2046	pounds	lb
tonne	metric tons (1000 kg)	2204.6	short tons	sh ton
<b>VOLUME</b>				
m <sup>3</sup>	cubic meters	35.234	cubic feet	cu ft
l	liters	1.0567	quarts	qt
cl	centiliters	0.10567	quarts	qt
ml	milliliters	0.03381	fluid ounces	fl oz
<b>TEMPERATURE (celsius)</b>				
°C	Celsius temperature	1.8	Fahrenheit temperature	°F

## Approximate Conversions from Metric Measures

Symbol	When You Know	Multiply by	To Find	Symbol
<b>LENGTH</b>				
in	inches	0.0254	meters	m
ft	feet	0.3048	meters	m
yd	yards	0.9144	meters	m
mi	miles	1.6093	kilometers	km
<b>AREA</b>				
sq in	square inches	6.4516	square centimeters	cm <sup>2</sup>
sq ft	square feet	0.0929	square meters	m <sup>2</sup>
sq yd	square yards	0.8361	square meters	m <sup>2</sup>
ac	acres	0.4047	hectares (100 000 m <sup>2</sup> )	ha
<b>MASS (weight)</b>				
oz	ounces	0.02835	grams	g
lb	pounds	4.5359	kilograms	kg
sh ton	short tons	907.18	metric tons (1000 kg)	tonne
<b>VOLUME</b>				
cu ft	cubic feet	0.02832	cubic meters	m <sup>3</sup>
qt	quarts	0.94635	liters	l
fl oz	fluid ounces	0.02957	liters	l
gal	gallons	3.7854	liters	l
cu yd	cubic yards	0.76456	cubic meters	m <sup>3</sup>
<b>TEMPERATURE (celsius)</b>				
°F	Fahrenheit temperature	0.5556	Celsius temperature	°C



© 1994 by The McGraw-Hill Companies, Inc. All rights reserved. Printed in the United States of America. This book is a trademark of The McGraw-Hill Companies, Inc.



## PREFACE

This is the third in a series of four reports prepared by International Computing Company (ICC) under Contract No. DOT-CG-81-78-1833, for the United States Coast Guard. This third report presents the software specifications and design for the Vessel Traffic Services (VTS) Processing/Display Subsystem.

The first report\* described the initial design study which focused on alternative architectures. The second report\*\* carried the initial design to a greater level of detail, discussed critical design issues and presented basic hardware specifications.

The fourth report\*\*\* is a companion volume to this report, providing a detailed design of an operating system which can support the VTS application in a multicomputer, high reliability environment.

This third report is essentially two reports in one. Part I presents the software specifications. Part II presents the current state of the software design. Both portions provide significant detail but leave open questions which cannot be answered at this time such as the precise characteristics of the display station hardware.

Both parts of this report begin with an overview and an introduction to the methodology used. The balance of both parts is a detailed, technically oriented, function by function description of the specification on the one hand and the design on the other.

\* Henson, C.C., Cleaver, R.A., Kaisler, S.H., "Preliminary Design Study for VTS Processing/Display Subsystem," June, 1978.

\*\* Henson, C.C., Mickey, F.T., Graham, R.S., McIntosh, B.A., "VTS Processing/Display Subsystem Design," January, 1979.

\*\*\*Cohn, D.A., Mickey, F.T., "VTS Processing/Display Subsystem Detailed Software Design - Operating System," July 1979.

A full understanding of the detailed material will require extensive and careful study. The reader should refer to the functional description\*\*\*\* prepared by the U.S. Coast Guard for the definition of terms not provided in this report.

Accession For	
NTIS GRA&I	<input checked="checked" type="checkbox"/>
DOC TAB	<input type="checkbox"/>
Unannounced	
Justification	
By _____	
Distribution/	
Availability	
Dist.	Avail and/or special
A	

\*\*\*\*VTS Processing/Display Subsystem Functional Description, Appendix 8, RFP 81-77-1833, September 1977.



PART II  
CONTENTS

<u>Section</u>		<u>Page</u>
1	INTRODUCTION	1-1
2	OVERVIEW OF VTS SOFTWARE DESIGN	2-1
2.1	Introduction	2-1
2.1.1	Display Station Processes	2-3
2.1.2	Processes in the Main Processor	2-3
2.2	Automatic Background Processing	2-5
2.2.1	Position Data Processing	2-5
2.2.2	Hazard Detection Processing	2-7
2.2.3	Post Alerts	2-16
2.2.4	Logging Management	2-17
2.2.5	Manage Environmental Sensors	2-17
2.3	Watchstander Interface Processes	2-18
2.3.1	Transact Watchstander Request	2-18
2.4	Watchstander Support Processes	2-21
2.4.1	Access Files	2-21
2.4.2	Demand Functions	2-25
2.5	Data Bases	2-28
2.5.1	Data Bases in Main Memory	2-28
2.5.2	Disc Resident Data Bases	2-29
2.5.3	Data Base Design Concepts	2-29
2.5.4	Access Methods	2-30
2.5.5	Files	2-33
3	AUTOMATIC BACKGROUND PROCESSING	3-1
3.1	Position Data Processing	3-2
3.1.1	CONVERT-RADAR-DATA Process	3-2
3.1.2	DISTRIBUTE-TRACKER-TABLES Process	3-25
3.1.3	MANAGE-POSITION-TABLES Process	3-29

PART II  
CONTENTS

<u>Section</u>		<u>Page</u>
3.2	Hazard Detection Processing	3-36
3.2.1	MONITOR-HAZARD-DETECTION Process	3-37
3.2.2	PREDICT-COLLISION Processes	3-43
3.2.3	DETECT-LANE-STRAY Process	3-56
3.2.4	DETECT-ROUTE-STRAY Process	3-62
3.2.5	PREDICT-GROUNDING Process	3-68
3.2.6	DETECT-EXCESSIVE-CONGESTION Process	3-74
3.2.7	PREDICT-DANGEROUS-ENCOUNTERS Process	3-80
3.2.8	DETECT-ANCHOR-DRIFT Process	3-93
3.2.9	DETECT-NAVAID-ADRIFT-MISSING Process	3-97
3.2.10	DETECT-EXCESSIVE-VESSEL-SPEED Process	3-102
3.3	Alert Posting	3-107
3.4	Logging	3-108
3.4.1	MAINTAIN-OPERATIONS-LOG Process	3-108
3.4.2	SAVE-MANUAL-BACKUP-DATA Process	3-116
3.4.3	MAINTAIN-TRAFFIC-SUMMARIES Process	3-121
3.4.4	MAINTAIN-PASSAGE-HISTORY Process	3-126
3.5	Environmental Sensors	3-129
4	WATCHSTANDER INTERFACE PROCESSING	4-1
4.1	TRANSACT-WATCHSTANDER-REQUESTS Process	4-2
4.2	MANAGE-MAP-DISPLAY Process	4-69
4.3	MANAGE-ALERT-DISPLAY Process	4-85
4.4	MANAGE-ACTION-REQUIRED-LIST (A-R-L) Process	4-97
5	WATCHSTANDER SUPPORT PROCESSING	5-1
5.1	Vessel File Access	5-2
5.1.1	Data Structures	5-3
5.1.2	ACCESS-VESSEL-FILE Process	5-9
5.1.3	MANAGE-VESSEL-INFORMATION-TABLE Process	5-30



PART II  
CONTENTS

<u>Section</u>		<u>Page</u>
5.2	Passage File Access	5-38
5.2.1	Data Structures	5-38
5.2.2	ACCESS-PASSAGE-FILE Process	5-40
5.3	Miscellaneous File and Table Access	5-51
5.3.1	Waterway File Access	5-51
5.3.2	Environmental File Access	5-73
5.3.3	Notices File Access	5-87
5.3.4	Display Station Status Access	5-94
5.3.5	Key Search	5-101
5.4	Demand Functions	5-106
5.4.1	DETERMINE-ENCOUNTERS	5-106
5.4.2	ACCESS-SYSTEM-PARAMETERS	5-110
5.5	Simulation	5-114
5.5.1	Data Structures	5-114
5.5.2	SET-UP-SCENARIO Process	5-115
5.5.3	PLAYBACK-SCENARIO Process	5-120
6	TESTING CONSIDERATIONS	6-1
6.1	Top-Down and Bottom-Up Testing	6-3
6.2	Testing During Development	6-4
6.3	Functional Testing	6-6
6.4	Load Testing	6-8
6.5	Performance Testing and Evaluation	6-10
6.6	Reconfiguration Testing	6-11
APPENDIX A.1	RELATIVE POSITION EQUATIONS	A-1
A.2	CPA EQUATIONS	A-3
A.3	DEAD-RECKONING COMPUTATIONS	A-5

PART II  
LIST OF FIGURES

<u>Figure</u>		<u>Page</u>
2-1	Major Groups of Processes	2-2
3-1	CONVERT-RADAR-DATA	3-3
3-2	Merged Vessel Pairs	3-16
3-3	DISTRIBUTE-TRACKER-TABLES	3-26
3-4	MANAGE-POSITION-TABLES	3-30
3-5	Position Table	3-32
3-6	MONITOR-HAZARD-DETECTION	3-38
3-7	PREDICT-COLLISION	3-44
3-8	DETECT-LANE-STRAY	3-58
3-9	DETECT-ROUTE-STRAY	3-64
3-10	PREDICT-GROUNDING	3-70
3-11	DETECT-EXCESSIVE-CONGESTIONS	3-76
3-12	PREDICT-DANGEROUS-ENCOUNTERS	3-81
3-13	PREDICT-DANGEROUS-ENCOUNTERS	3-82
3-14	DETECT-ANCHOR-DRIFT	3-94
3-15	NAVAID-ADRIFT-MISSING	3-98
3-16	DETECT-EXCESSIVE-VESSEL-SPEED	3-103
3-17	MAINTAIN-OPERATIONS-LOG	3-109
3-18	SAVE-MANUAL-BACKUP-DATA	3-117
3-19	MAINTAIN-TRAFFIC-SUMMARIES	3-122
3-20	MANAGE-ENVIRONMENTAL-SENSORS	3-130
4-1	TRANSACTION-WATCHSTANDER-REQUESTS	4-3
4-2A	MANAGE-MAP-DISPLAY	4-70
4-2B	MANAGE-MAP-DISPLAY	4-71
4-2C	PROCESS-MESSAGE	4-72
4-3A	MANAGE-ALERT-DISPLAY	4-86
4-3B	MANAGE-ALERT-DISPLAY	4-87
4-4A	MANAGE-ACTION-REQUIRED-LIST	4-98
4-4B	MANAGE-ACTION-REQUIRED-LIST	4-99



PART II  
LIST OF FIGURES

<u>Figure</u>		<u>Page</u>
5-1	Multi-Level Index Structure	5-4
5-2	Vessel Record Data Structures	5-8
5-3A	ACCESS-VESSEL-FILE	5-10
5-3B	ACCESS-VESSEL-FILE	5-11
5-4	SETUP-FOR-INDEX-SEARCH Variables	5-26
5-5	MANAGE-VESSEL-INFORMATION-TABLE	5-31
5-6	Passage File Index Organization	5-39
5-8A	ACCESS-PASSAGE-FILE	5-43
5-8B	ACCESS-PASSAGE-FILE	5-44
5-9A	Waterway Data Base Description	5-55
5-9B	Waterway Cell Data	5-56
5-10A	ACCESS-WATERWAY-DATA	5-59
5-10B	ACCESS-WATERWAY-DATA	5-60
5-11	UPDATE-WATERWAY-FILES	5-67
5-12	Environmental File Records	5-74
5-13A	ACCESS-ENVIRONMENTAL-DATA	5-76
5-13B	ACCESS-ENVIRONMENTAL-DATA	5-77
5-14A	ACCESS-NOTICES-FILE	5-88
5-14B	ACCESS-NOTICES-FILE	5-89
5-15A	ACCESS-DISPLAY-STATION-STATUS	5-95
5-15B	ACCESS-DISPLAY-STATION-STATUS	5-96
5-16	SEARCH-ON-KEY	5-103
5-17	DETERMINE-ENCOUNTERS	5-107
5-18	System Parameter List	5-111
5-19	ACCESS-SYSTEM-PARAMETERS	5-112
5-20	SET-UP SCENARIO	5-117
5-21	PLAYBACK-SCENARIO	5-121

The design of a software system involves the translation of a set of software requirements into a set of program modules, tables and data bases which, when implemented, will perform the required functions. The design process is essentially the selection of techniques, structures, algorithms, etc., from nearly infinite possibilities.

Design is a complex process which is not yet completely understood. Great strides have been made, however, in developing concepts, techniques and tools which can assist the designer in doing his job. These techniques allow the designer to approach his task in an orderly fashion.

In addition to providing some assistance to the designer, recently developed techniques make what may well be a more significant contribution by providing techniques for communicating a design to others so that they can understand clearly what the designer intended. This understanding makes it possible for others to evaluate, critique, and, if need be, alter the design.

In developing and describing the design of the software for the VTS Processing/Display Subsystem, we have used a number of techniques and concepts which should be noted before the design is presented.

At the highest level of the design, we have organized the software as a group of cooperating sequential processes. This concept



was discussed in the design report\* which preceded this document.

Each process is designed as a hierarchy using top-down design techniques.\*\* Top-down design leads to well structured, modular software.

Three techniques have been used to describe the design. Initially, ordinary language is used to describe the processes and their interaction. In some cases, this description alone is sufficient. In other cases, particularly in those cases involving significant complexity, a Program Design Language (PDL) description and/or a structure chart\*\* has been used to describe the design.

PDL is a kind of structured English. A number of PDL's have been suggested in the literature. The PDL we have used is based on PASCAL control structures. PDL is a program-like description of a program but is a more simplified, less detailed, more readily understood representation than the program itself.

In addition to PDL, we have used structure charts to provide a diagrammatic description of the control flow of particular processes.

These concepts and techniques have all proved to be helpful to us in designing and describing our design. We trust that those who will later concern themselves with this design will also find the techniques helpful, at least in understanding the design in its current state.

\* Henson, C.C., Mickey, F.T., Graham, R.S., McIntosh, B.A., "VTS Processing/Display Subsystem Design," January, 1979.

\*\* Myers, G.J., "Reliable Software Through Composite Design," Petrocelli/Charter, New York, 1975.



## 2.1 INTRODUCTION

At the highest level of the design, the software for the VTS Processing/Display Subsystem has been decomposed into processes which operate concurrently and cooperate to perform the required functions. Processes will communicate and synchronize their activities by the exchange of explicit messages and answers.\* This approach provides flexibility in assigning processes to processors, which is a critical factor in providing systems that can manage a broad range of functional and load handling requirements. A simplified overview of the software design is shown in Figure 2-1. Included are major groups of processes and the major interprocess communications paths. The configuration assumed will handle the majority of VTS sites. Groups of processes are assigned to either a display station processor or to the main system processor. Additional processors will be included in the largest VTS configurations which will require further separation of the processes.

A discussion of the processes required can be organized in a number of ways. In the simplified overview, processes are grouped based on their assignment to processors. Following that, a more extensive overview will be based on the organization of the remainder of this design document.

\*For further discussion of processes and the concepts on which the design is based see: Henson, C. C., Mickey, F. T., Graham, R. S., McIntosh, B. A.; VTS Processing Display Subsystem Design, Chapters 7 and 8, January, 1979.

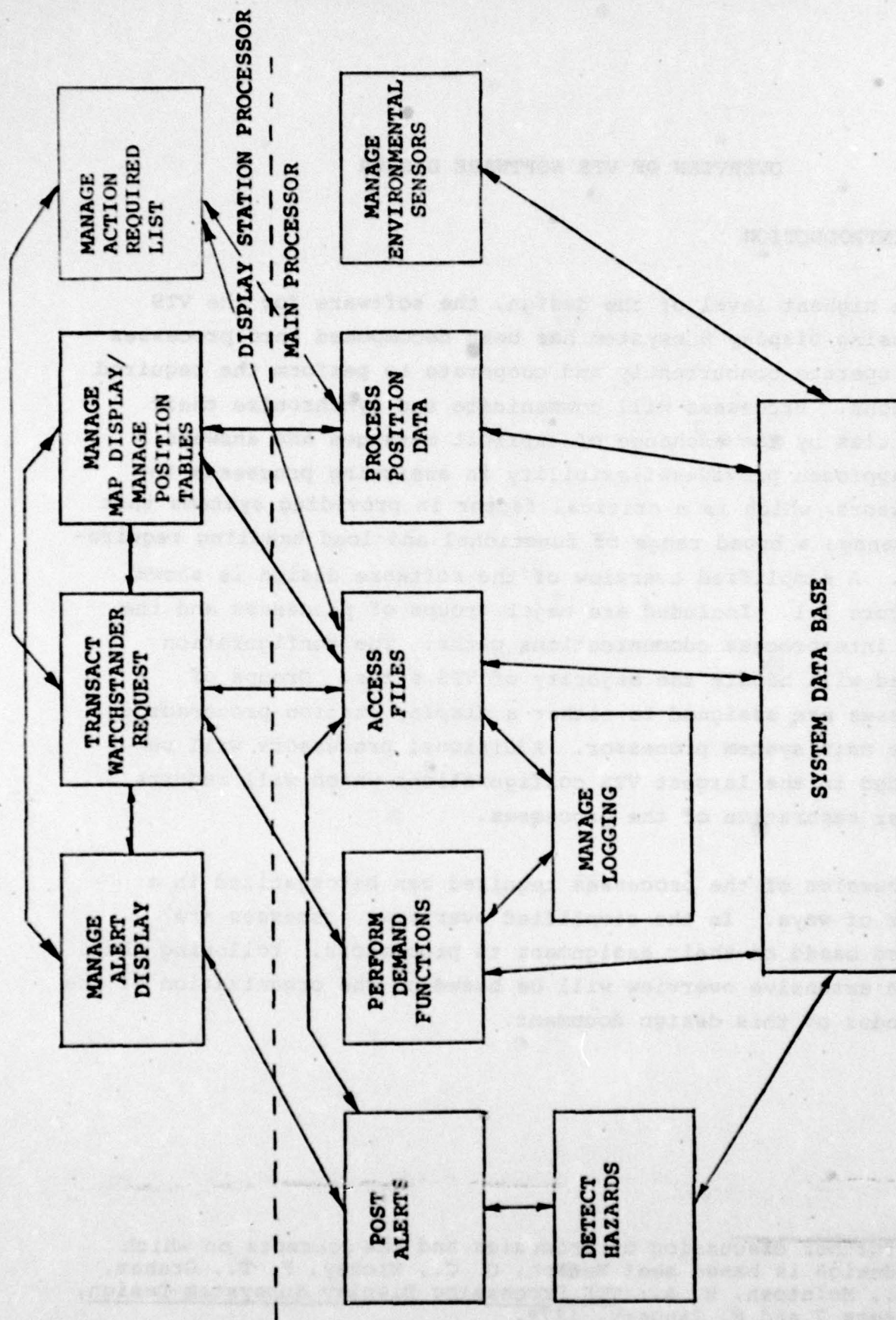


Figure 2-1. Major Groups of Processes



### 2.1.1 Display Station Processes

The display station processor will include four major processes. One of these processes, Transact Watchstander Request, will provide the primary interface between the system and the watchstander. It will accept inputs from the watchstander and respond to those inputs. If the function cannot be handled entirely by the Transact Watchstander Request process, messages will be sent to other processes which can perform the required operation.

A second major process, Manage Map Display, will manage the map display. It will communicate with a number of other processes which will supply up-to-date information to be displayed.

The other two major processes, Manage Alert Display and Manage Action Required List, will manage the alert display and a list of other items requiring the attention of the watchstander, respectively. The Manage Alert Display process will receive messages from the Post Alerts process in the main processor. It will also receive messages from the Transact Watchstander Request process. The Manage Action Required List process will receive its messages from processes, such as the position data processes, to insert elements in the list and from the Transact Watchstander Request process to remove elements from the list.

### 2.1.2 Processes in the Main Processor

A variety of processes will be required in the main processor to support the functions of the VTS Processing/Display Subsystem. These processes (i.e., service processes) will support the watchstanders by performing demand functions and file accesses.

In a VTS configuration that includes automatic position sensors, such as tracking radar (Level 4 sensors), a group of processes will be required to receive the inputs from the position sensors, convert the data as required, store the converted data and distribute the data as needed to the map display processes.

One of the larger group of processes will perform hazard detection. One or more processes will be needed for each type of potential hazard to be monitored. Hazard detection processes will send information to the Post Alerts process when an alert condition is detected.

Other processes will be required to manage environmental sensors and manage logging.

In our simplified diagram (see Figure 2-1), we have also shown a common data base which is accessed by a number of processes. The common data base allows one process to compile information, which is subsequently used by another process, without any direct communication between the process which gathers the data and the process which uses it.



## 2.2 AUTOMATIC BACKGROUND PROCESSING

A number of processes will operate in the background of the watchstander invoked functions. These functions interface with radar and other sensors and monitor the state of the waterway to detect potentially hazardous conditions. From the standpoint of the computer system, these functions can be considered the most critical since they require a major part of the system resources.

### 2.2.1 Position Data Processing

Three processes, or families of processes, have been identified to perform the functions related to position sensors, including the processing required to support tracking radar. They are Convert Radar Data, Distribute Tracker Tables, and Manage Position Tables, and are described in the following subsections.

#### 2.2.1.1 Convert Radar Data

A family of interrupt level processes will be provided to interface with radar tracking units. These processes will receive the data as it is transmitted by the radar units and queue the data for processing by the Convert Radar Data process.

The Convert Radar Data process will perform coordinate conversions and update the main memory tables which contain current location, course and speed of all tracked vessels. Status information included with the radar input will be checked for error, dropped contact, new contact, or merged contact flags.

The Convert Radar Data process will attempt to match new contacts with vessels already known to the system, such as a vessel entering the radar area from a Level 1 area. Merged tracks will also be processed by keeping a record of merged vessels and attempting to match them with new contacts.

When the Convert Radar Data process has completed conversion, storage and analysis of the data, a message will be sent to the Distribute Tracker Tables process for distribution to display processors.

#### 2.2.1.2 Distribute Tracker Tables

The Distribute Tracker Tables process will receive messages from the Convert Radar Data process and send update messages to appropriate manage map processes.

If the interprocessor bus supports broadcast messages, the Distribute Tracker Tables process can send a single message to all processors simultaneously. If a broadcast capability is not available, the Distribute Tracker Tables process will send update messages in sequence to each appropriate manage map process.

#### 2.2.1.3 Manage Position Tables

The Manage Position Tables process handles access to the position tables. It receives messages from other processes requesting current position data. An answer message will be formulated including the requested data.

This process is used to provide information hiding and flexibility. However, critical hazard detection processes will bypass the Manage Position Table process and use the data directly to reduce the overhead associated with the use of position data.



### 2.2.2 Hazard Detection Processing

Processes will be included to attempt to detect potentially hazardous situations and provide information (alerts) which will allow the vessel or vessels involved to be notified so the hazard can be avoided.

The VTS Processing/Display Subsystem will have the capability of detecting the following hazards:

- . Potential Collision
- . Lane Stray
- . Route Stray
- . Potential Grounding
- . Excessive Congestion
- . Dangerous Encounters
- . Anchor Drift
- . Navaid Adrift/Missing
- . Excessive Vessel Speed

The task of the software is to recognize these conditions and report the potential hazards to the watchstanders as alerts.

In order to recognize these conditions, the software must monitor the data on Nav aids and the location, course, and speed of vessels.

The frequency of execution of the hazard monitoring processes will determine the amount of system resources required to perform the functions. An increased frequency of execution may be desirable but must be traded-off against the increased use of system resources. The VTS Processing/Display Subsystem has, therefore, been designed to handle a maximum frequency of execution for each



hazard detection process. The Watch Supervisor will be allowed to adjust the time cycles, so long as he does not request greater than the maximum allowable frequency (See Section 2.4.2.4, Access System Parameters).

Ideally, the subsystem would generate an alarm only when a truly hazardous condition exists. A variety of normal conditions can, however, be cited which would result in an alarm. To prevent these false alarms the subsystem will also allow the Watch Supervisor to exempt any particular physical area(s) from any or all of these hazard checks. This capability may be used, for example, to prevent false collision warning alerts in narrow channels where vessels pass so close together that even in a normal passing situation an alert would result.

Additionally, the subsystem will allow the Watch Supervisor to exempt particular vessels from any or all of the hazard checks. This capability may be used to avoid false collision alarms for pilot boats, tugs, etc., which have intentional "collisions" as a part of their every day operation, or to prevent lane stray alarms for ferries, etc., which are not normally constrained to lanes.

#### 2.2.2.1 Hazard Detection Monitor

The Hazard Detection Monitor process will control the time schedule for the hazard detection processes. It will base its functions on the time schedule set by the Watch Supervisor and send messages to trigger individual hazard detection processes. When each process completes its cycle, it will send an answer to the monitor process which will record its completion.

If processes fail to complete in the scheduled time, the monitor process will generate error messages indicating that the system has degraded. If the subsystem is operating in a degraded

mode, the Watch Supervisor will have the option of adjusting the time schedule or the relative priorities assigned to each type of hazard detection process.

#### 2.2.2.2 Predict Collisions

The subsystem will be designed to detect potential collisions at sites equipped with sensors which can determine the location, course and speed of the vessels with a high degree of accuracy.

The detection of potential collisions will require a periodic check on each pair of vessels in the area covered by Level 4 or 5 sensors. Depending on the algorithm used, the processing required can be very large since for  $n$  vessels,  $n(n-1)/2$  checks must be made.

To minimize the processing required, three conditions which are successively more selective have been specified. A separate process will be used to monitor each of these three conditions.

##### Stage 1 Process

The Stage 1 process will check each vessel pair in the VTS coverage area every 30 seconds (minimum cycle time). The Stage 1 process is initiated by a message from the Hazard Detection Monitor process. The first step in the Stage 1 processing is to formulate a collision table. The collision table will contain vessel position, course and speed data from the tracker tables. Vessel data is extracted from the tracker tables so that only vessels need be examined.

An efficiently coded loop will be used to compare the coordinates of each pair of vessels. If the difference in the X or Y coordinates is less than the selectable tolerance, additional checking will be performed to determine if the vessel pair is exempt from collision



processing. A vessel pair will be exempt if neither is identified, both are anchored or docked, one or both vessels has been marked exempt, one or both vessels is in a designated exempt area, or one or both vessels is not in track.

If the vessel pair is not exempt, it will be entered into the Stage 2 list for further checking by the Stage 2 process.

### Stage 2 Process

The Stage 2 process checks each vessel pair entered in its list by the Stage 1 process every 15 seconds (minimum cycle time). For each vessel pair in the list, the most recent radar input data will be used to calculate Closest Point of Approach (CPA) and time to CPA. If CPA and time to CPA are both less than the values set by the Watch Supervisor, the vessel pair will be entered into the Stage 3 list for checking by the Stage 3 process.

### Stage 3 Process

The Stage 3 process will analyze vessel pairs which have been entered into its list by the Stage 2 process. The minimum cycle time will be 6 seconds. Processing includes an estimation of collision risk based on:

- . CPA
- . The rate that the CPA is increasing or decreasing (dCPA)
- . The time remaining until CPA is reached (tCPA)
- . The approximate sizes of the vessels involved (one of four sizes for each vessel)
- . The relative hazard of the cargos aboard.

The risk will be estimated using these factors in the following equation:

$$\text{Risk} = C_4 \frac{(S - \text{CPA})}{t\text{CPA}} - C_5 \frac{d\text{CPA}}{t\text{CPA}} + H$$

where  $C_4$  and  $C_5$  are weighting constants for their associated factors. "S" is the vessel size risk factor which has one of 10 values based on a table of risk values for the ten possible combinations of the four vessel size categories. "H" is the risk value based on the hazard of the cargo. Up to 20 cargo types will have values associated with them in the data base. "H" will equal the value of the most hazardous cargo aboard a vessel, or if both vessels are carrying hazardous cargo, it is the sum of the values for the most hazardous cargo aboard each vessel. The tables of size and cargo risk values will be data base constants accessible by the Watch Supervisor.

The third stage process will generate an alert message to the Post Alerts process when the risk value exceeds a threshold set by the Watch Supervisor and has exceeded it for more than an adjustable time period. The time limit required before an alert is issued will reduce the probability of false alerts being generated. When the risk value drops below the established threshold for longer than the time limit, the third stage process will send a cancellation message to the Post Alert process.

#### 2.2.2.3 Detect Lane Stray

The Detect Lane Stray process will cycle once every 30 seconds (minimum cycle time). All vessels which are in track by Level 4 or 5 sensors and are following a lane will be checked to determine if the vessel is outside the proper lane of travel or will be outside the lane in "n" minutes based on straight line dead reckoning. The constant "n" will be a data base constant accessible by the Watch Supervisor.

If the vessel is outside the assigned lane or will be, an alert message will be sent to the Post Alert process.



As with other hazard detection processes, the Detect Lane Stray process will be initiated by a message from the Hazard Detection Monitor process. When it has completed its cycle it will send an answer to the monitor process.

#### 2.2.2.4 Detect Route Stray

The Detect Route Stray process will cycle at a rate not faster than once every 30 seconds. It will be initiated by a message from the Hazard Detection Monitor process and will generate an answer when it completes its cycle. Each identified non-exempt vessel's reported location will be compared with its intended route. A route stray alert will be issued if a vessel is in track by Level 4 or 5 sensors and the reported position is not within a predefined distance from any route segment of a vessel's prescribed route.

A route stray alert will also be generated by the Access Passage File process (see Section 2.4.1.2) when an Update Vessel Position message is received indicating that a vessel within a Level 1, 2 or 3 area is not on its intended route. This type of route stray alert will not in any way affect the Detect Route Stray process.

#### 2.2.2.5 Predict Grounding

The Predict Grounding process will check each identified vessel for potential grounding at a rate not faster than once every 30 seconds. The potential grounding check will be based on:

- . The reported draft of the vessel
- . The depth of water at Mean Lowest Low Water (MLLW) in the route segments or cells through which the vessel is expected to pass
- . The present water level relative to MLLW from water level sensors, tide schedules or manual entry.

The potential grounding detection process will dead reckon each vessel ahead up to a preset amount of time (as selected by the Watch Supervisor but not longer than 10 minutes) and verify that the actual water level on its intended route over that time period (i.e., the depth at MLLW plus the height of water above or below MLLW) exceeds the draft of the vessel. If a vessel is following a route structure, this dead reckoning will be based upon its reported speed and its intended route, and will consider the depth of the appropriate route segments. If, however, the vessel is not following a route structure, the dead reckoning will revert to straight line dead reckoning based on the vessel's current course and speed, and will use for the grounding determinations the depth of that portion of the waterway which the dead reckoned track will traverse.

If grounding is projected on the current course of the vessel, an alert message will be sent to the Post Alerts process.

When each cycle of the grounding detection process is completed, a completion message will be sent to the Hazard Detection Monitor process. The Predict Grounding process will then wait for another message from the monitor process before repeating the cycle of grounding checks.

#### 2.2.2.6 Detect Excessive Congestion

The Detect Excessive Congestion process will detect when more vessels are in or will be in a critical area than the area can safely accommodate. The checks will be repeated at a rate not faster than once every 30 seconds. Up to 40 critical areas will be defined by a center point and a radial distance (in a two dimensional harbor representation) or by waypoints and linear distances along routes from waypoints (in a one dimensional representation). Each critical area will be assigned a maximum



capacity value. The amount of actual congestion will be determined using the technique defined below which yields a congestion value. If this value will exceed the critical area's maximum capacity within a pre-established look-ahead time span (not greater than 30 minutes); a congestion alert message will be sent to the Post Alerts process.

The look-ahead time span will be a data base constant accessible by the Watch Supervisor. The look-ahead will be accomplished by advancing all vessels which are in a route structure along their intended route at their reported (or measured) speed. Vessels outside a route structure will not be considered. The value each vessel contributes toward this total capacity will be dependent upon its size and cargo risk value. A table of capacity contribution constants for each of four vessel size categories will be stored in the data base and be accessible by the Watch Supervisor station. The capacity contribution resulting from the various hazardous cargos will be the same as the 20 values established for these cargos in the potential collision hazard detection process (see Section 2.2.2.2). The actual congestion of an area will be the sum of the size and cargo capacity contribution constants from each vessel in the critical area.

As with the other hazard detection processes, the Detect Excessive Congestion process will be initiated by a message from the Hazard Detection Monitor process. When it has completed its checks it will send an answer to the Hazard Detection Monitor process.

#### 2.2.2.7 Predict Dangerous Encounters

The Predict Dangerous Encounters process is used to detect when vessels are approaching a passing or overtaking situation in a channel too narrow to safely accomplish it, or when a vessel will



be crossing the channel too close to an oncoming vessel in the channel. The location of each endpoint of up to 20 constricted areas will be specified in the data base. At a preset time interval before a vessel will enter any constricted area, the subsystem will verify that the constricted area will be free of opposing traffic of sufficient size to be hazardous. Also, whenever a vessel is about to cross the channel, the movement will be accomplished at least a preset amount of time before or after any vessel in the channel arrives at the crossing location.

Each of the preset times mentioned above will be data base constants accessible to the Watch Supervisor. Also in the data base for each constricted area are specifications of the size combinations of vessels (one of four sizes for each vessel) which constitute a dangerous encounter. The Predict Dangerous Encounters process will cycle at a rate not faster than once every 30 seconds. The rate, determined by the Watch Supervisor defined data base value, will be controlled by the Hazard Detection Monitor process.

#### 2.2.2.8 Detect Anchor Drift

The Detect Anchor Drift process will check the present measured position of all identified anchored vessels within the coverage of Level 4 or 5 sensors. An anchor drift alert message will be sent to the Post Alerts process if a vessel moves outside the specified swing radius from its assigned anchorage location. This process cycles at a rate not faster than once per minute.

#### 2.2.2.9 Detect Navigational Aid (Navaid) Adrift or Missing

Each navaid designated by the Watch Supervisor, which is within the coverage area of Level 4 or 5 sensors, will be compared to its normal position. The normal display symbol for a navaid is

replaced with a special symbol when Level 4 or 5 sensors lose track of a specified navaid (navaid missing), or if a navaid in track is outside its specified watch circle (navaid adrift). When a missing or adrift condition is detected a message will be sent to the Post Alerts process which will in turn transmit a message to the Manage Alert Display process in the appropriate display station processors. It will inform the Manage Map process of the need for a special symbol. Processing to detect a navaid adrift/missing will be repeated at a rate not faster than once every minute, as determined by the message from the Hazard Detection Monitor process.

#### 2.2.2.10 Detect Excessive Vessel Speed

The Detect Excessive Vessel Speed process will check the measured speed of vessels in all route segments and "cells" for which a maximum speed has been specified. If a vessel is proceeding at a speed "x" knots above the limit (where "x" is a data base constant accessible by the Watch Supervisor), a message will be sent to the Post Alerts process. Speed limit checks will be performed at a rate not faster than once every minute and will be controlled by messages from the Hazard Detection Monitor process.

#### 2.2.3 Post Alerts

The Post Alerts process receives messages from the hazard detection processes described in Sections 2.2.2.2 to 2.2.2.10. Alert messages will be forwarded to appropriate Manage Alert Display processes in the display station processors. The Post Alerts process will be responsible for monitoring a Master Alert Queue.



#### 2.2.4 Logging Management

Logging management will provide support for the logging functions carried out by the subsystem. A family of processes will be used, with one member supporting each type of log.

The logging management processes will receive messages from processes wishing to enter data into a particular log. The logging management process required will then enter the data into a buffer and call the operating system to write the data to the logging medium when the buffers are full.

#### 2.2.5 Manage Environmental Sensors

The Manage Environmental Sensors process will receive data from environmental sensors and enter the data into the system data base.



## 2.3 WATCHSTANDER INTERFACE PROCESSES

Four major application processes will provide the interface between the computer system and the watchstanders. From the standpoint of the computer system, these processes are less critical than the background processes since they use fewer of the total system resources. As a part of an overall VTS system, however, the computer system should serve as a tool for the watchstanders. The design of the interface processes will determine to a great extent how effective a tool the Processing/Display Subsystem will be and how easily it can be adapted to revised requirements which will almost certainly develop as the system is used.

These processes which include Transact Watchstander Request, Manage Map Display, Manage Alert Display and Manage Action Required List will be supported by processes in the main or other processors (see Section 4, Watchstander Interface Processing).

### 2.3.1 Transact Watchstander Request

One member of the Transact Watchstander Request family of processes will reside in each of the display station processors. By way of the operating system, Transact Watchstander Request receives inputs from the watchstander's standard and function keyboards. Following the entry of a function key or function identifier, the function will be validated and the appropriate procedure will be called to handle the specified function.

The particular procedure will output appropriate forms and/or cues to the watchstander. Entries by the watchstander will be received and verified. Whenever required, the Transact Watchstander Request process sends messages to other processes in the main processor. It then waits for the function, such as a data base inquiry, to be performed and an answer sent. It then formats and outputs the appropriate information.

When the procedure is completed, the process again waits for an input of a function key or function identifier from the watchstander.

A partial list of the functions which the Transact Watchstander Request must support includes:

1. Enter/Modify/Delete/Display Vessel Record
2. Enter/Modify/Delete/Display Passage Record
3. Update Vessel Position
4. Enter New Communication
5. Identify Vessel
6. Modify Checkpoint
7. Change Status
8. Enter/Modify/Delete/Display Route Segments
9. Enter/Modify/Delete/Display Cell Data
10. Enter/Modify/Delete/Display Notice
11. Enter/Modify/Delete Manual Environmental Data
12. Enter/Modify/Delete Forecast
13. Environmental Information Display
14. Enter/Modify Delete Simulation (Watch Supervisor only)
15. Initialize/Start/Stop Simulation Playback (Watch Supervisor only)
16. Key Search
17. Local Traffic
18. Encounters
19. Relative Position
20. Closest Point of Approach
21. Schedule /Route
22. Adjust System Parameters (Watch Supervisor only).



Recognition of function keys and function codes and selection of the appropriate procedures to handle the functions will use CASE statements which are PASCAL equivalents of a table driven program. This will make it easy to add, modify or delete functions.

## 2.4 WATCHSTANDER SUPPORT PROCESSES

The final major category of processes to be considered are those that primarily support the watchstander by providing the services which Transact Watchstander Request will need. These processes will manage access to major files and perform a variety of demand functions.

These support processes will be designed as functional elements with each process handling a separate function or a closely related group of functions. One or a family of processes will be assigned to manage accesses to each major file or to perform a major function. This functional design of processes will, we believe, reduce the complexity of the software and simplify the task of utilizing additional processors for the larger VTS configurations.

### 2.4.1 Access Files

Service processes will be provided in the main processor to perform display (retrieve), enter, modify, and delete record functions for the system files. For nearly all the system files a single process is more than adequate to handle the number of accesses to be handled per second, even in the larger configurations. However, it may be desirable to provide two or more processes for those functions associated with the passage file in the larger, more demanding VTS configuration.

#### 2.4.1.1 Access Vessel File

The Access Vessel File process will receive messages from the Transact Watchstander Request process (Section 2.3.1). These messages will instruct the Access Vessel File process to enter a new vessel record, retrieve a vessel record for display, retrieve a vessel record for potential modification, replace a vessel record with an updated copy, or to delete a vessel record. These operations will support the display, enter, modify and delete vessel functions.



The Access Vessel File process will maintain a list of vessel records which are in use by each watchstander. This will allow a watchstander to be notified in the unlikely event that another watchstander is already using the record when a request is made. A watchstander can also be notified if another watchstander has altered the record during the time he was attempting to alter it.

The Access Vessel File process will maintain the indexes to the vessel file. The vessel file will contain a fixed maximum number of records. The Access Vessel File process will be responsible for assigning unused records when a new vessel is entered. If the file is full, the process will select a vessel which has not been accessed recently, automatically delete that record and assign the record space to the new vessel.

When the process has completed the requested operation it will send an answer to the requesting process indicating that it was successful or unsuccessful and, if unsuccessful, the reason. If a retrieval was requested and was successful, the retrieval record will also be forwarded to the requesting process.

#### 2.4.1.2 Access Passage File

The Access Passage File process performs essentially the same function as the Access Vessel File process except that it deals with the passage file rather than the vessel file. It receives messages from the Transact Watchstander Request process which instructs it to enter a new passage record, retrieve a passage record for display, retrieve a passage record with intent to update, replace a passage record with an updated copy, or to delete a passage record.

These operations will support the passage file functions listed below.

1. Enter Passage
2. Modify Passage
3. Delete Passage
4. Display Passage
5. Update Vessel Position
6. Enter New Communication
7. Identify Vessel
8. Modify Checkpoint
9. Change Status

The Access Passage File process will manage the allocation of free space in the passage file and assure that the most critical vessels are included in the passage file by automatically deleting less critical vessels if need be to make space available. Normally, however, space should be available since space is made available regularly when vessels leave the coverage area. In addition, the passage file will be sized to handle more than the maximum anticipated number of vessels.

The Access Passage File process will maintain the indexes to the passage file and will maintain a list of watchstanders who are modifying passage records so that notification can be given when needed.

When the process has completed the requested operations, it will send an answer to the requesting process indicating that it was successfully or unsuccessfully completed. If a passage record was successfully retrieved, it will be included with the answer.

#### 2.4.1.3 Search on Key

The Search on Key process receives input messages from the Transact Watchstander Request process (Section 2.3.1). A message



will include the identification of the file to be searched and the logical combination of keys to be used. For example, the message might request a search of the vessel file for all vessels over 300 feet long which have been in the VTS coverage area within the last month.

The Search on Key process will scan the specified file and create an answer which includes those records which match the search criteria. The search operation will read large blocks of the file and perform a sequential search of each block.

#### 2.4.1.4 Access Environmental Data

The Access Environmental Data process will receive input messages from the Transact Watchstander Request process (Section 2.3.1) and from the Manage Environmental Sensors process (Section 2.2.4). It will control the entry, updating and retrieval of both manually entered and automatically collected environmental data.

#### 2.4.1.5 Access Waterway Data

The Access Waterway Data process will receive messages from the Transact Watchstander Request process (Section 2.3.1). Messages will include requests to enter, retrieve, replace, or delete records of cell data, notices, route descriptions, waypoints, docks and piers, navigational aids and other data related to the characteristics of the waterway. A single process is required to handle these functions since they are infrequently utilized.

The Access Waterway Data process will perform the requested operations and formulate an answer which will report success, report errors or return the requested data.

#### 2.4.2 Demand Functions

In addition to the functions described in Section 2.4.1 which enter and retrieve information from the system, other functions will be provided which perform calculations, analysis and other processing based on information stored in the VTS Processing/Display Subsystem.

If the information is readily available in the display station processor, the Transact Watchstander Request process (Section 2.3.1) will perform the function itself. CPA calculations and relative position determinations, for example, can normally be done by the Transact Watchstander Request process. Other functions will be performed by individual processes in the main processor.

##### 2.4.2.1 Get Local Traffic

The Get Local Traffic process receives messages from the Transact Watchstander Request process (Section 2.3.1). A message includes a vessel and a radius. An answer is formulated which includes all vessels within the specified distance from the vessel.

The Get Local Traffic process will examine the current position of each vessel known to the system. The algorithm used for Stage 1 of Predict Collision processing (Section 14.2.1.2, Part I) should be used to make a rapid check of the differences in x and y coordinates. This effectively screens out all vessels which are not within a square area surrounding the designated vessel. An actual distance calculation can then be used for vessels which are not rejected by the initial screening operation.

##### 2.4.2.2 Determine Encounters

The Determine Encounters process receives a message from the Transact Watchstander Request process which includes a vessel and a look-ahead time span. The location, course, and speed of other vessels within the VTS coverage area will be compared against that of the specified vessel. An



answer will then be formulated which includes all vessels which the specified vessel will meet, cross or overtake during the look-ahead time.

#### 2.4.2.3 Route/Schedule

The Route/Schedule process will receive an input message from the Transact Watchstander Request process specifying a vessel identifier and proposed route and schedule. It will formulate an answer indicating that the proposed route or schedule is acceptable or that potential problems have been detected. If problems are detected, the subsystem will attempt to adjust the schedule or select an alternate route which will be included with the answer.

In a large VTS configuration, a substantial amount of data may have to be analyzed to perform the routing and scheduling function in spite of the fact that no elaborate optimization procedures will be required. If this function is used extensively, a family of processes will be needed to allow multiple vessels to be scheduled nearly simultaneously. For the largest of the configurations we have analyzed, a processor and a disc are effectively dedicated to this function.

#### 2.4.2.4 Access System Parameters

The Access System Parameters process will receive messages from the Transact Watchstander Request process which is interacting with the Watch Supervisor. The messages may include a designation for the parameter(s) to be changed and the new value(s) to assign. The parameters are used to define and adjust the operation of the system. Included are parameters which define the rate at which hazard detection processes operate and parameters which define constricted areas.

To allow maximum flexibility , an extensive set of options will be available to the Watch Supervisor. Since a considerable amount of software may be needed for this process and since only one individual at a time may be using these functions, it may be appropriate to overlay the programs which support this process.

When the function has been completed, an acknowledgment message (answer) will be returned to the Transact Watchstander Request process.

#### 2.4.2.5 Setup Scenario

The Setup Scenario process will receive messages from the Transact Watchstander Request process requesting that the appropriate functions be performed. The function will allow a new scenario to be setup, real data to be recorded in the scenario and records for the simulated data base to be entered, modified or deleted.

Simulation will be used for training operations personnel and for exercising and testing the system under artificial conditions.

Answers will be returned to the Transact Watchstander Request process indicating the completion of the requested operation and including retrieved simulated data when appropriate.

#### 2.4.2.6 Playback Scenario

An additional process will be included to control the playback of a simulation scenario. The scenario data will be read from the data bases and appropriate data will be sent to other processes to allow the playback process to operate as a source for messages normally generated by other operational processes.



## 2.5 DATA BASES

Information storage and retrieval are central to the VTS Processing/Display Subsystem. A portion of the information is automatically acquired by sensors, such as radar; however, the balance of information is entered by watchstanders.

A portion of the data requires extremely rapid access and will be stored in main memory. Moving head discs will be used to store data which is less frequently accessed or must be permanently recorded.

### 2.5.1 Data Bases in Main Memory

A number of data bases in the form of tables and other data structures will be kept in the main memory. Most of these tables and data structures will contain information that is created and used by a single process.

Two major tables have been identified, however, which are of a more global nature. One of these two tables, the Tracker Table, will contain information from the radar tracking units. It will be updated by the Convert Radar Data process and data from the table will be used by a number of other processes, including the hazard detection processes. The Tracker Table will be kept in main memory because it must be accessed hundreds of times a second to update location, course and speed and to retrieve this information for hazard detection.

A second major table, the Vessel Information Table, will contain additional information needed frequently for hazard detection. It will include the size, draft and cargo hazard value for each active vessel. This information will be used by Predict Collision, Predict Grounding and other hazard detection processes.

Two indexes will also be held in main memory. These indexes allow conversion from one means of identifying a vessel to another. One index gives the internal vessel ID as a function of a track and tracker number. The other index is the converse.

Copies of these tables and indexes may be needed in more than one processor since processes which use them heavily may exist in more than one processor. In smaller VTS configurations which would not automatically require additional copies, at least two processors should have up-to-date copies so that a processor failure does not destroy critical data. Since the radar data regenerates itself every six seconds, it would not be a critical loss. The loss of the indexes which establish the correspondence between vessel identities and tracker data could, on the other hand, necessitate reidentification of all vessels by the watchstander.

#### 2.5.2 Disc Resident Data Bases

Vessel, passage, environmental and waterway characteristics data will be stored on moving head disc files. Coupled with the indexes necessary to access the data, these disc files will form the major disc resident data bases.

The term "disc resident data bases" has been used to distinguish them from the data bases in main memory. Because this terminology is cumbersome, the remaining discussion will simply refer to "data bases" to denote disc resident data bases.

#### 2.5.3 Data Base Design Concepts

The data base design emphasizes both simplicity and efficiency because of the real-time nature of the system and because of the response time requirements which the system must meet. We have chosen not to use a generalized data base management system (DBMS) because of the overhead associated with separating the



physical and logical data files. Instead, the DBMS functions have been tailored to VTS by making the logical and physical files identical and by designing data access methods appropriate for each file, taking into consideration the number of keys utilized, as well as response time requirements.

Maximum retrieval efficiency is not the primary goal. The VTS data base consists of portions which are relatively static. The most heavily utilized files, however, are dynamic. Records are added, deleted and modified frequently. Therefore, access mechanisms must be designed to provide satisfactory response times for all these operations.

#### 2.5.4 Access Methods

A number of techniques have been considered for accessing files in the VTS Processing/Display Subsystem. The technique we suggest for files which cannot be addressed directly is based on a multi-level index structure. This access mechanism is straightforward and its performance is easily predicted. It also is suitable for files from which records are frequently deleted. Some other access methods, such as hashing, are excellent for efficient retrieval but are less desirable when records must be deleted.

A multi-level index structure is shown in Figure 2-2. At each level of index, the keys will be in ascending alphanumeric order. At all levels except the lowest level, the entries will indicate the first key in that block of keys. By comparing successive pairs of entries at the first level, the block containing the next (or last) index level can be identified. For all levels except the lowest, each entry will contain the key and a pointer to the beginning of the next block of keys. At the lowest level, the index entry will contain the key and the location of the desired record.

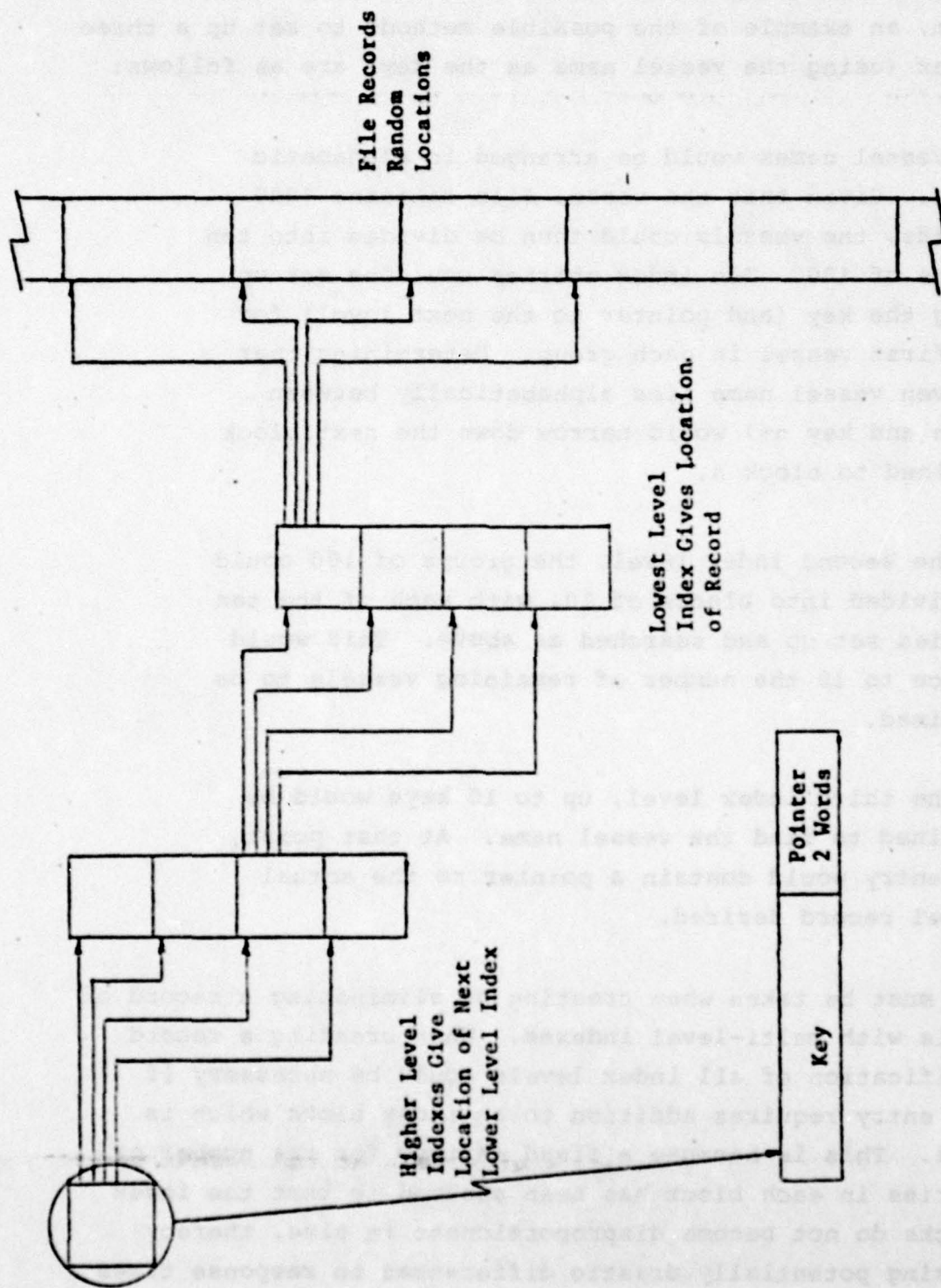


Figure 2-2. Multi-Level Index Structure



To illustrate the multi-level index structure proposed for the VTS system, an example of the possible methods to set up a three level index (using the vessel name as the key) are as follows:

- . The vessel names would be arranged in alphabetic order. Given that the vessel file contains 1000 records, the vessels could then be divided into ten groups of 100. Ten index entries could be set up using the key (and pointer to the next level) for the first vessel in each group. Determining that a given vessel name lies alphabetically between key  $n$  and key  $n+1$  would narrow down the next block examined to block  $n$ .
- . At the second index level, the groups of 100 could be divided into blocks of 10, with each of the ten entries set up and searched as above. This would reduce to 10 the number of remaining vessels to be examined.
- . At the third index level, up to 10 keys would be examined to find the vessel name. At that point, the entry would contain a pointer to the actual vessel record desired.
- . Care must be taken when creating or eliminating a record of a file with multi-level indexes. When creating a record, modification of all index levels could be necessary if the entry requires addition to an index block which is full. This is because a fixed maximum for the number of entries in each block has been assumed so that the index blocks do not become disproportionate in size, thereby causing potentially drastic differences in response times for data access. When deleting a record, similar modifications to the indexes will be required if deleting the record causes an index block to be emptied.

For access to non-indexed files, or for searches on a key or keys which are not supported by indexes, the entire subject file will be examined to find the particular record or records of interest or to satisfy the requirements of the particular search.

#### 2.5.5 Files

A number of files will be used to store the records which make up the VTS data bases. These files will contain information on the waterway included in the VTS coverage area, vessels which have been, are or will be in the coverage area, and information on passages which vessels are currently making through the coverage area. Other files will hold map outlines and other formats for display. Also included are the system software, and other support files.

Major files will be described briefly in the subsections that follow. More detailed descriptions are provided in Section 5 of this report and are associated with the descriptions of the software which access these files.

##### 2.5.5.1 Vessel File

The vessel file contains information about vessels which does not normally change from one passage to the next. A vessel record will include a vessel identification, e.g.:

- . Name
- . Lloyd's Registry or Military Hull Number
- . Radio Call Sign

Indexes will be provided for all three identifiers so that any one identifier will be sufficient to locate a desired vessel record.



A vessel record will also contain data which is characteristic of the vessel itself including:

- . Type
- . Gross weight
- . Maximum and minimum draft
- . Beam, length and height at minimum draft
- . Maximum speed
- . Time to crash stop
- . Number of screws
- . Type of navigational equipment aboard
- . Minimum turning radius
- . Doctor aboard normally
- . Distance to crash stop

The following information about the owner and agent will be included in the vessel record:

- . Flag
- . Owner
- . Name, address and phone number of local agent

In addition, a vessel record will confirm status, linkage and miscellaneous information including:

- . Date data last verified
- . Date vessel last active
- . Miscellaneous comments
- . Links to associated records

#### 2.5.5.2 Passage File

The passage file contains data concerning vessels which are in or about to enter the VTS coverage area. A passage record will be created when a vessel is imminent (i.e., about to enter the VTS coverage area) and will be deleted when the vessel exits from the coverage area.

A passage record will include the vessel name, type and ID code. It will also contain current information about the vessel which will vary from one passage to the next, i.e.:

- . Pilot name and designation
- . Barge makeup
- . Cargo
- . Actual draft

The passage file will also include data on the vessel's route and schedule:

- . Origin or point of entry to VTS coverage area
- . Destination or planned point of exit
- . Date/time of entry
- . Scheduled date/time of exit
- . Intended route
- . Lane program flag

A passage record will also include linkage to the associated vessel record and to records of communications between the vessel and the watchstander. In addition, a passage record will contain a status indicator that will specify that the passage is imminent, underway, anchored or docked.

If the status is imminent, only the data discussed above will be included in the passage record. For an underway vessel, however, a record of checkpoints passed will be included in the record if checkpoints are valid, i.e., outside tracker ranges. Of particular importance is the identity of the last checkpoint passed and when it was passed, the next checkpoint expected and the estimated time of arrival at the next checkpoint.



For anchored vessels the following data will be included:

- . Designation of the anchorage
- . Swing radius of mooring
- . Location of anchorages
- . Date/time anchorage established
- . Date/time of scheduled departure
- . Intended Route

A lesser amount of information is needed for docked vessels.

A passage record for a docked vessel will include the dock or pier designation, Date/Time Arrived, Date/Time of scheduled departures and Intended route.

Three indexes will be maintained to allow rapid access to the passage file:

- . Vessel ID code
- . Vessel name
- . Pilot designation

#### 2.5.5.3 Waterway Characteristics File

Several record types will be maintained in the Waterway File to define the geographic characteristics of the waterway and significant features in the waterway such as Nav aids, docks and piers.

Five separate data structures are now envisioned which include:

- . Route segments
- . Waypoints
- . Docks and Piers
- . Nav aids
- . Waterway data by cells (small sized grid)

#### 2.5.5.4 Environmental Data Files

The Environmental Data file will contain records containing information, as a function of location within the waterway, about:

- . Weather status, such as temperature, and visibility, from automatic and manual input;
- . Waterway status, such as direction and speed of current, and tide level;
- . Weather forecast.

Four specific environmental data structures have been identified including:

- . Automatic weather station data
- . Automatic current tide data
- . Forecasts
- . Environmental data manually entered.

#### 2.5.5.5 Other Data Structures

A number of other data structures will also be included to hold notices to Mariners and map display data such as copies of outlines in multiple magnifications. Other data structures will also be included such as software files and miscellaneous temporary support files.



Automatic background processing includes position data processing, hazard detection, posting of alerts to the display station and logging. Background processes are initiated automatically, rather than by watchstander requests. Some background processes are initiated (directly or indirectly) by external device interrupts, such as the arrival of data from radar trackers. Other background processes, such as the hazard detection processes, are automatically invoked at preset time intervals.

### 3.1 POSITION DATA PROCESSING

This section describes the processes which collect, convert, and distribute vessel and buoy position data input from radar and/or the watchstander. The processes are CONVERT-RADAR-DATA, DISTRIBUTE-TRACKER-TABLES and MANAGE-POSITION-TABLES. CONVERT-RADAR-DATA collects radar data, converts it to a standard format, and sends it to DISTRIBUTE-TRACKER-TABLES which sends the new position data to MANAGE-POSITION-TABLE or MANAGE-MAP-DISPLAY in each processor.

#### 3.1.1 CONVERT-RADAR-DATA Process

**Brief Description:** This process collects the radar data from each radar station, one track at a time, and prepares a record of the new entries and/or changes to be made in the position table in each processor. These records are sent to the DISTRIBUTE-TRACKER-TABLES process (see Section 3.1.2).

There is a copy of the CONVERT-RADAR-DATA process in each of the processors which are connected to the radar stations; however, only one is active at any time.

CONVERT-RADAR-DATA control/data paths are shown in Figure 3-1.

**Relationship to Functional Description:**

Appendix 8, Section 4.2(d)

**Changes from Appendix 8:**

An overlap flag has been included in the radar data.

This flag is set whenever the tracked object is in an area covered by more than one radar.



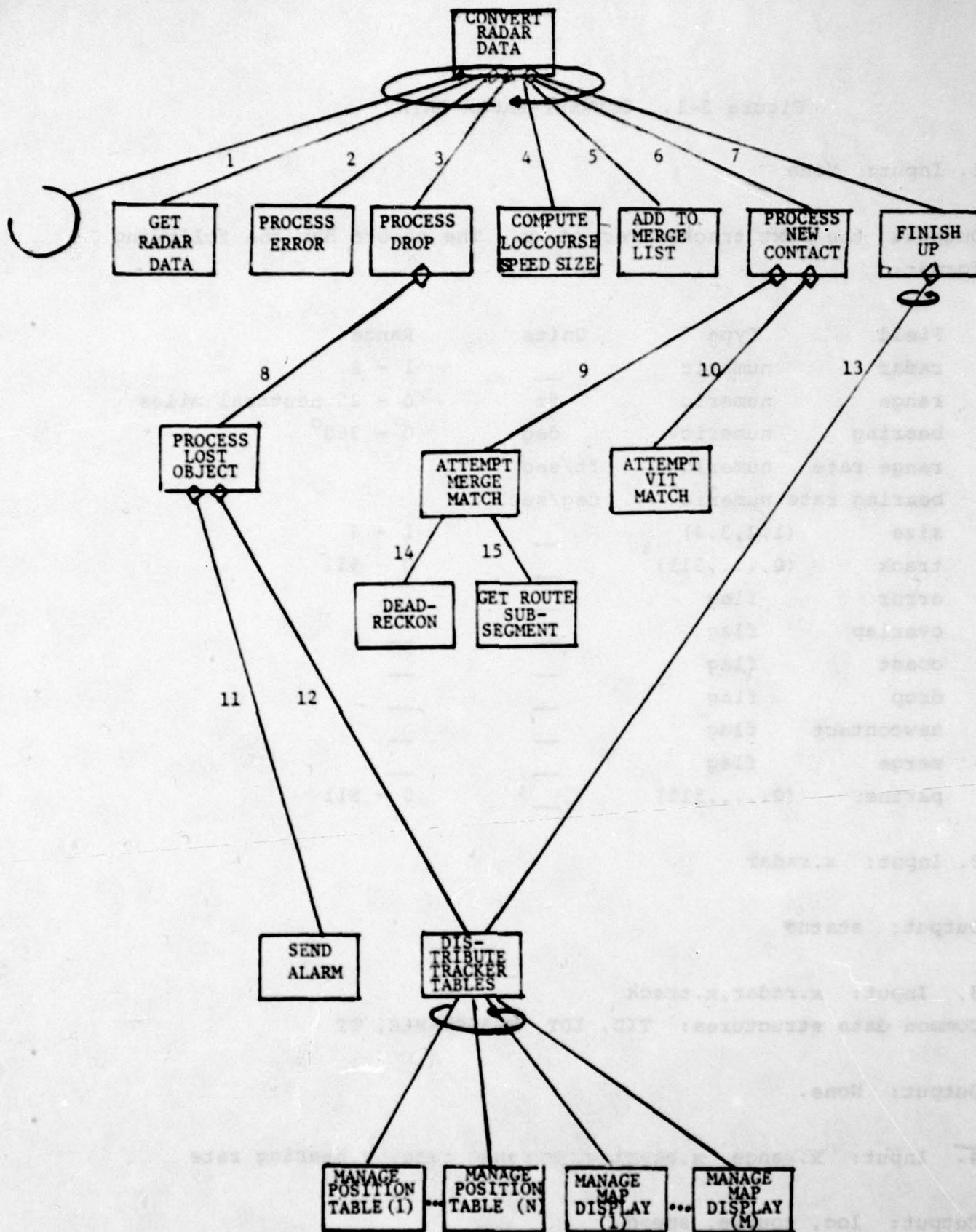


Figure 3-1. CONVERT-RADAR-DATA

Figure 3-1. CONVERT-RADAR-DATA

1. Input: None

Output: the next tracker record, x. The record has the following format:

Field	Type	Units	Range
radar	numeric	—	1 - 8
range	numeric	ft	0 - 25 nautical miles
bearing	numeric	deg	0° - 360°
range rate	numeric	ft/sec	
bearing rate	numeric	deg/sec	
size	(1,2,3,4)	—	1 - 4
track	(0,...,511)	—	0 - 511
error	flag	—	—
overlap	flag	—	—
coast	flag	—	—
drop	flag	—	—
newcontact	flag	—	—
merge	flag	—	—
partner	(0,...,511)	—	0 - 511

2. Input: x.radar

Output: status

3. Input: x.radar,x.track

Common data structures: TID, IDT, MERGETABLE, TT

Output: None.

4. Input: x.range, x.bearing, x.range rate, x.bearing rate

Output: loc, course, speed



5. Input: x.radar,x.track,x.partner

Common data structure: TID, IDT, Merge Table, TT

Output: None

6. Input: x.overlap, loc, course, speed, x.size

Common data structures: TT, TID, Merge Table; IDT, Overlap List

Output: None.

7. Input: x.radar, x.track, loc, course, speed, x.size

Common data structures: TT, IDT, Merge Table

Output: none

8. Input: ID

Common data structures: IDT, Merge Table, TT

Output: None

9. Input: loc, course, speed, size, TT (m.radar, m.track),  
merge table entry m

Common data structures: IDT, TID

Output: successful flag

10. Input: loc, course, speed, x.size

Common data structure: VIT (see Section ), TID, IDT

Output: successful

11. Input: ID

Output: None

12. Input: ID, 'untracked'

Output: none

13. Input: ID, loc, course, speed, size  
Output: none

14. Input: m.loc, m.course, m.speed, m.time  
Output: deadreckoned.location

15. Input: ID  
Output: current and next route segment



### Frequency of Use:

#### Class C

#### Class B

#### Class A

#### Expected:

Full scan (up to 512 tracks)

Same as C

0

is sent for each radar once

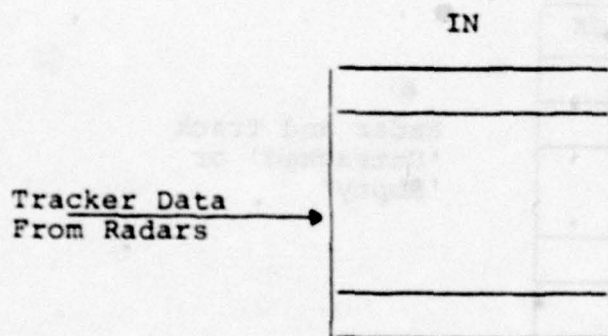
0

every six seconds.

Peak: Same as Expected

### Data Structures:

IN - Buffer of messages from the radar stations that one track of information is ready.



IDT - This table connects the internal identification of an object with the radar and track number currently being used to track the object. If more than one radar is tracking the object, then an attempt is made to choose the "best" radar; i.e., if one radar is tracking it as a single object and a second radar is tracking the object as part of a merged pair, then the first radar is chosen.

It is possible to have vessels in the system which are untracked. The IDT entry for these vessels is 'untracked.' If an internal ID is not connected with a tracked object or a vessel in the Passage File, then the IDT entry is 'empty.' The empty entries may be linked together.

The ID values will be used to distinguish among identified vessels, unidentified vessels and bouys. For ease of discussion 0-3999 have been chosen for the identified vessels, 4000-7999 for the unidentified vessels and 8000-10000 for the bouys. In actual implementation the values should be parameters.

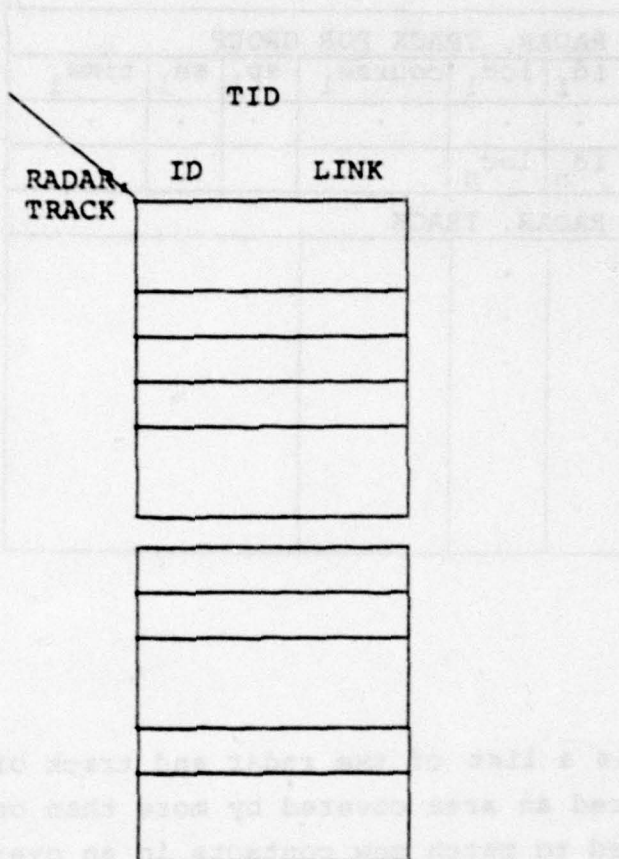
IDT		
ID	RADAR	TRACK
0	RADAR	TRACK
17		
130		
10,000		

Radar and Track  
'Untracked' or  
'Empty'

TID - This table connects the radar and tracker number with the internal ID. If the radar track represents a single object (i.e., the merge flag has not been set for this track since the last time the new contact flag was set), then the entry will be the internal identification of the object and a link. The link field will be empty if the object is not being tracked as a single object by any other radar.



If the radar track represents more than one vessel, then the TID entry will be a pointer to a Merge Table entry representing all the objects.



Merge Table - A merge table entry consists of two parts. The first part is the radar and track being used to track the group. The second part contains information about all the vessels in the group. This information is the internal ID of the vessel; the location, course, speed and size at the time the object became merged; and the time when the object became merged. This information will be used to deadreckon the vessel to separate the vessels after they become unmerged.

# MERGE TABLE

RADAR, TRACK FOR GROUP					
id <sub>1</sub>	loc <sub>1</sub>	course <sub>1</sub>	sp <sub>1</sub>	se <sub>1</sub>	time <sub>1</sub>
.	.	.	.	.	.
id <sub>n</sub>	loc <sub>n</sub>				
RADAR, TRACK					
	.				
	.				
	.				

Overlap - This is a list of the radar and track of the objects which have entered an area covered by more than one radar. The list will be used to match new contacts in an overlap area with other objects in the area. When a match is found the object will be linked in the TID table and removed from the Overlap list. It may be necessary to purge old entries if the list becomes long.

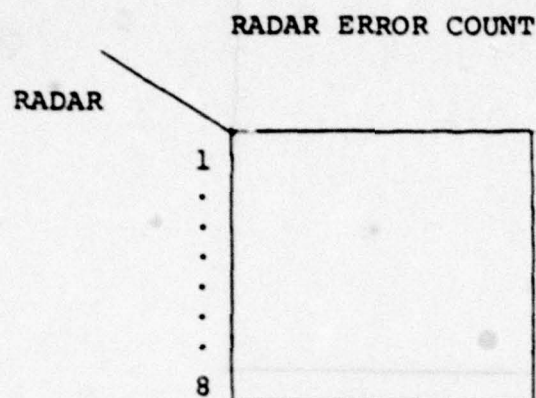
## OVERLAP

RADAR	TRACK

A list of  
vessels in  
overlap area



Radar Error Count - This is used to total the number of errors reported by each radar.



TT - This table has the last error-free location, course, speed and size reported for each track. The information will be recorded in the form most useful to the processes which need this data. (See below)

dlat = integer; difference in lat, lon from center point of VTS area, See Appendix 8, page 8-68.

dlon = integer;

```
coordinate = record
              one: dlat
              two: dlon
              end,
```

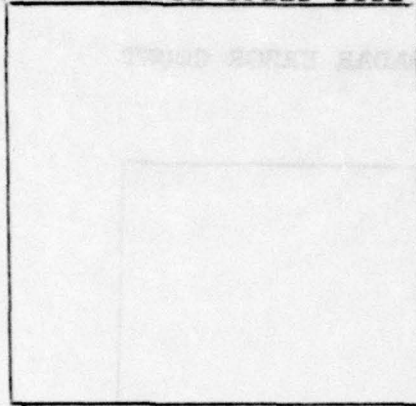
```
trackrecord = record
              location: coordinate;
              course: real;
              speed: real;
              size: 1..4;
              end;
```

TT = array (1..n radar, 0..511) of track record;

TT

RADAR, track  
1, 0

LOC COURSE SPEED SIZE



8,511



Process

CONVERT-RADAR-DATA

begin

while active

do

if no radar data on IN list

then

WAITMESSAGE;

GET-RADAR-DATA(X);

if x.error

then PROCESS-ERROR(x.radar);

else

if x.dropped

then PROCESS-DROP;

else

do COMPUTE-LOC-COURSE-SPEED-SIZE;

if x.overlap and not x.merge

then put x.radar, x.track on overlap list;

if x.merge

then ADD-TO-MERGE-LIST;

if x.newcontact

then PROCESS-NEW-CONTACT;

FINISH-UP; .

end,

end;

end;

end.

## Procedures

### 1) GET-RADAR-DATA

This procedure returns the next tracker record. See Section 3.1.1.2 for the description of the tracker data.

### 2) PROCESS-ERROR (x.radar)

begin

if error count (x.radar) > tolerance

then

do switch to other radar line;

    alert operator of radar line failure;

end;

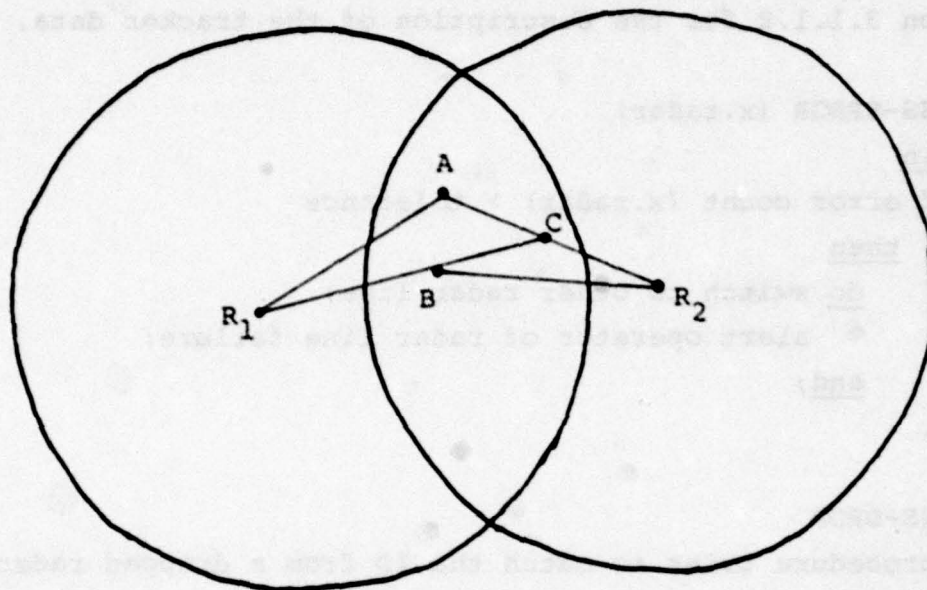
end.

### 3) PROCESS-DROP

This procedure tries to match the ID from a dropped radar track with another radar track if the vessel is tracked by more than one radar station. There are several cases to consider since one radar may be tracking two or more vessels as a merged group while another radar considers the two as separate vessels (See Figure 3-2).

If no other radar is tracking the vessel, the object is dropped from the display if there is no passage record. If the vessel has a passage record, the vessel is dead-reckoned. In addition, an alarm is sent to the operator if the vessel should still be within the radar area.





Radar1 is tracking A as a single vessel  
and B and C as a merged pair

Radar2 is tracking B as a single vessel  
and A and C as a merged pair

Figure 3-2. MERGED VESSEL PAIRS

PROCESS-DROP

begin

case TID (x.radar,x.track)

ID, link:

if link is not empty /\*another radar is tracking the  
single vessel\*/

then do

IDT(ID):=link;

remove TID(x.radar,x.track) from  
the link chain; end;

else /\*no radar is tracking this vessel as a single  
vessel but it may be part of a merged group\*/

PROCESS-LOST-OBJECT(ID);

merge table pointer:

if the radar and track stored in the merge table entry  
is equal to x.radar,x.track

then /\*radar has lost the merged group which was  
being tracked on x.radar,x.track\*/

do remove the merge table entry;

for each ID in the merge group

if IDT(ID) = x.radar,x.track

then PROCESS-LOST-OBJECT(ID);

end;

end;

TID(x.radar,x.track): = blank;

TT(x.radar,x.track): = blank;

end.



4) COMPUTE-LOC-COURSE-SPEED-SIZE

This procedure uses the bearing, range, and rates to produce location, course, speed and size.

5) ADD-TO-MERGE-LIST

This procedure creates a merge table entry for all IDS associated with x.radar,x.track or x.partner (the merged radar,track).

begin

Get an empty space in the merge table for new merge entry  
the radar,track for new entry is x.radar,x.track;

for K:= x.radar,x.track and x.partner

do case TID(K)

    ID,link:

do if link  $\neq$  ^

then do

                IDT(ID): = link;

                /\*this keeps ID associated with a non-merged  
                pair if possible\*/

                remove TID(K) from link chain; end;

else

            IDT(ID): = x.radar,x.track;

            Add ID,TT(K) and current time to new  
            merge table entry;

end;

merge table entry;

do for each ID in old merge entry

do if IDT(ID) = the radar, track in  
        old merge entry

then IDT(ID): = x.radar,x.track;

        move the ID, loc course speed, size,  
        time into new merge entry; end;

remove old merge entry from merge table

end;

```

        TID(K): = pointer to new merge entry;
    end;
end.

```

6) PROCESS-NEW-CONTACT

```

    begin
        found:=false;
        if x.overlap /*check other objects in overlap areas*/
        then
            while found = false and not at the end of overlap list
            do Get next o.radar,o.track from overlap list
                if computed loc speed size are close to
                    TT(o.radar,o.track)
                then do
                    TID(x.radar,x.track): = TID(o.radar,o.track);
                    link TID(x.radar,x.track);
                    remove x and o from overlap list;
                    found: = true;
                end;
            end;
        while found = false and not at end of merge Table
        do Get next merge entry m
            if computed loc course speed size is close to
                TT(m.radar,m.track)
            then
                do if only two ID in merge group
                    then do ATTEMPT-MERGE-MATCH
                        IF successful
                            then do found: - true;
                    end
                remove merge table entry end;

```



```

        else do found: = true; /* to stop further
                                processing*/
                                Get new unidentified vessel ID;
                                TID(xradar,x.track): = 'ID;
                                IDT(ID): = x.radar;
                                end;
        end;

    While found = false
    do ATTEMPT-VIT-MATCH;
    if not successful
    then do Get new unidentified vessel ID;
        TID(x.radar,x.track): = ID;
        IDT(ID): = x radar;
    end;
end.

```

# 7) FINISH-UP

```

    begin
    TT(x.radar,x.track):= computed loc course speed size
    case TID(x.radar,x.track)
    ID, link:
        if IDT(ID) = x.radar,x.track
        then send ID, and TT(x.radar,x.track) to
            DISTRIBUTE-TRACKER-TABLES;
    merge entry:
        for all ID in merge table entry
        if IDT(ID) = x.radar,x.track
        then send ID, TT(x.radar,x.track)
            to DISTRIBUTE-TRACKER-TABLES;
    end.

```

8) PROCESS-LOST-OBJECT

This procedure attempts to match a dropped single vessel (or one vessel of a dropped merged group) with a merged group tracked by another radar.

begin

Search merge table identifiers;

if one of the merged identifier matches ID

then /\*another radar is tracking it as part of a merged group\*/

IDT(ID): = radar, track stored in the merge entry;

else /\*no radar is tracking the single vessel\*/

if there is passage record for the vessel

then

do Send TT(x.radar,x.track) to MANAGE-VIT  
for deadreckoning.

IDT(ID): = 'untracked';

Send (ID, 'untracked') to DISTRIBUTE-TRACKER-TABLES;

if vessel is not near the edge of radar area

then SEND-ALARM;

end;

else

Add ID to free list;

Send (ID, 'empty') to DISTRIBUTE-TRACKER-TABLES;

end.

9) ATTEMPT-MERGE-MATCH

Input: TT(m.radar,m.track)

computed loc,course,speed,size /\* latest tracker data

merge table entry corresponding to m



Processing: An attempt is made to match m and n with the ID1 and ID2 sorted in the merge table entry. If the match can be made with confidence (defined below), then match is recorded in the TID and IDT tables. For example if n corresponds to ID1 and m corresponds to ID2

```
then  TID(n): = ID1;
      TID(m): = ID2;
      IDT(ID1): = n;
      IDT(ID2): = m;
      successful: = true;
```

#### Function

##### TEST-OF-CONFIDENCE

```
if the difference in sizes of m and n is greater than 2
  then if the two sizes store in the merge entry are the same
    as the sizes of m and n
      then the match can be made with confidence based on
        the sizes;
    else;
  else
    do
      DEAD-RECKON both of the merge table entries;
      if the deadreckoned locations course speed and size of
        ID1 and ID2 are close to the location course speed
        and size of m and n
        then
          if the difference in course of m and n is greater
            than 45 degrees
            then then match can be made with confidence
          else
            do GETSUBSEGMENT(ID1);
                GETSUBSEGMENT(ID2);
            if the locations of m and n are on the
                route subsegments
            then
              if the route subsegments are not the
                same
```

then the match can be made with  
confidence

else

do send a message to Acknowledgment  
Required;

TID(n) := an unidentified ID;

TID(m) := an unidentified ID;

IDT(ID1) := \*\*; /\* \*\* implies  
that the system  
cannot connect the  
vessel with a  
track \*/

IDT(ID2) := \*\*; /\* this will stop  
all further  
attempts by the  
system to identify  
n \*/

end



10) ATTEMPT-VIT-MATCH

Input: the computed loc, course, speed, size

Processing: searches the VIT table to see manually entered data matches the location course speed and size.

begin

if the match is very close

then do TID(n):= the id matched;

IDT(id):=n;

a message is sent to the map process

to put a special symbol on the map

for this vessel; /\* since it has been

automatically identified by system\*/

successful:= true;

end;

else do

if match is close /\*but not with 90% condidence\*/

then send a message to Acknowledgment Required;

successful:= false;

end.

11) SEND-ALARM

Notifies the operator of identified vessels which have disappeared unexpectedly from the radar by sending an Enter-Alert message to the POST-ALERTS process.

### 3.1.2 DISTRIBUTE-TRACKER-TABLES Process

**Brief Description:** This process is responsible for distributing the position updates to all the processors which need position data. In order to minimize bus communication overhead, the process collects several messages from the FINISH-UP and PROCESS-LOST-OBJECT processes before sending them out on the bus.

It is also responsible for reporting to the Main Error Reporting Center (MERC) all delays by the other processors in accepting the data.

DISTRIBUTE-TRACKER-TABLES control/data paths are shown in Figure 3-3.

#### **Data Structures:**

**IN** - a buffer for collecting the position updates from the PROCESS-LOST-OBJECT and FINISH-UP processes.

**SEENDBUFFER** - a buffer for accumulating several updates to be sent at one time.



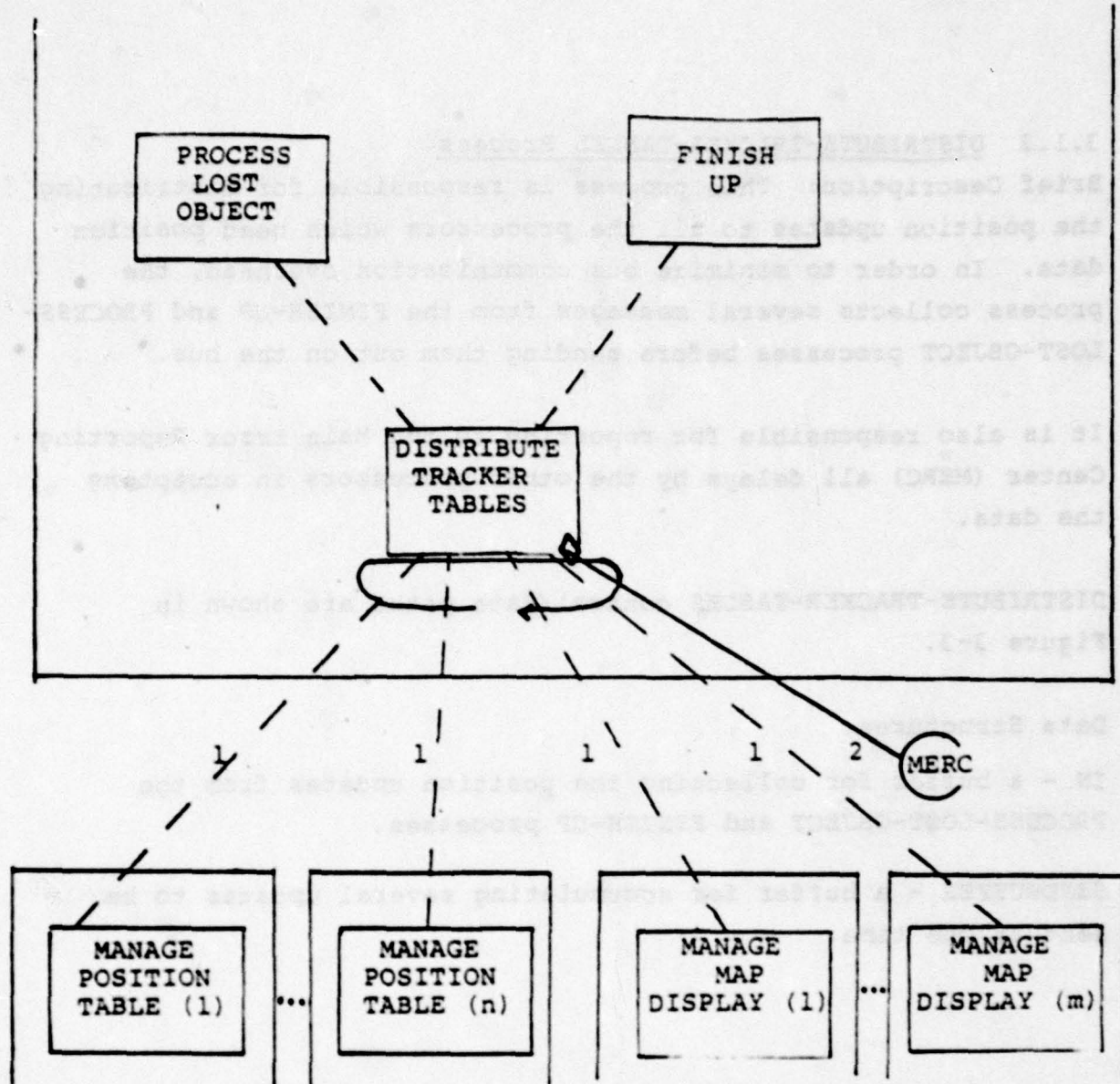


Figure 3-3. DISTRIBUTE-TRACKER-TABLES

Figure 3-3. DISTRIBUTE-TRACKER-TABLES

- 1 Input: A buffer of position messages. There are three kinds of messages

- ID, LOC, COURSE, SPEED
- ID, 'untracked'
- ID, 'empty'

Output: Status message that the data has been accepted.

- 2 Input: Name of the process which has not answered. (The delay may be due to a communication problem or an overloading problem within the processor containing the process)

Output: The output will depend on the MERC's analysis of the problem.

If the process cannot be contacted, (i.e., a break in the communication line, failed processor, etc.) then the MERC will generate an artificial answer to the message that was sent to the failed process. All further messages to the process will be artificially answered. When the problem is repaired, a copy of the current position table and Vessel Information Table (VIT) will be loaded.

If the problem is due to overloading, the MERC can delay the DISTRIBUTE-TRACKER-TABLES until the process can answer.



Process DISTRIBUTE-TRACKER-TABLES

begin

start timer; /\*This will guarantee that a position update gets  
sent after a period of time, even if the buffer  
is not full\*/

while active

do

while SENDBUFFER is not full and timer not finished

do if there is no message on the IN list

then WAITMESSAGE,

Put message in SENDBUFFER;

end

if SENDBUFFER isn't empty

then

do

SENDMESSAGE (Process 1);

. to all processes manage  
. position data.

SENDMESSAGE (Process n);

SENDMESSAGE (CLOCK); /\*This will guarantee that the

process is awakened, even if

one of the processors has failed\*/

while all of the processes have not reported

do WAITANSWER;

if the answer is for the clock

then report the problem to MERC;

end

Mark SENDBUFFER as empty;

end;

Start timer;

end;

end.

### 3.1.3 MANAGE-POSITION-TABLES Process

**Brief Description:** MANAGE-POSITION-TABLES is the process containing the two procedures responsible for inserting radar data into and retrieving radar data from the Position-Table.

There is a copy of both procedures in each processor which handles position data. The procedures are slightly modified in the processors containing the CONVERT-RADAR-DATA process (see note under Figure 3-4.)

MANAGE-POSITION-TABLES control/data paths are shown in Figure 3-4.

#### **Data Structures:**

**INDEX:** An array which matches an internal ID with the address of the corresponding radar data in the Position-Table. The first 4000 entries are reserved for IDs of vessels with passage records. The next 400 entries are reserved for IDs of vessels and objects without passage records. The last 2000 entries are for buoys.

During implementation the number of each type of vessel and the number of buoys should be an adjustable parameter. For this discussion, however, these numbers are represented as constants in order to simplify the discussion.

The INDEX array makes it possible to keep the radar data on each type of vessel and on each bouy with without leaving extra space in the Position-Table or rearranging the Position-Table if the number of tracked vessels or bouys changes.

**POSITION-TABLE:** An array containing the location, course, speed and size of all tracked objects. It will contain an 'untracked' flag if the corresponding vessel is being manually updated or dead reckoned.

Figure 3-5 shows the Position-Table.



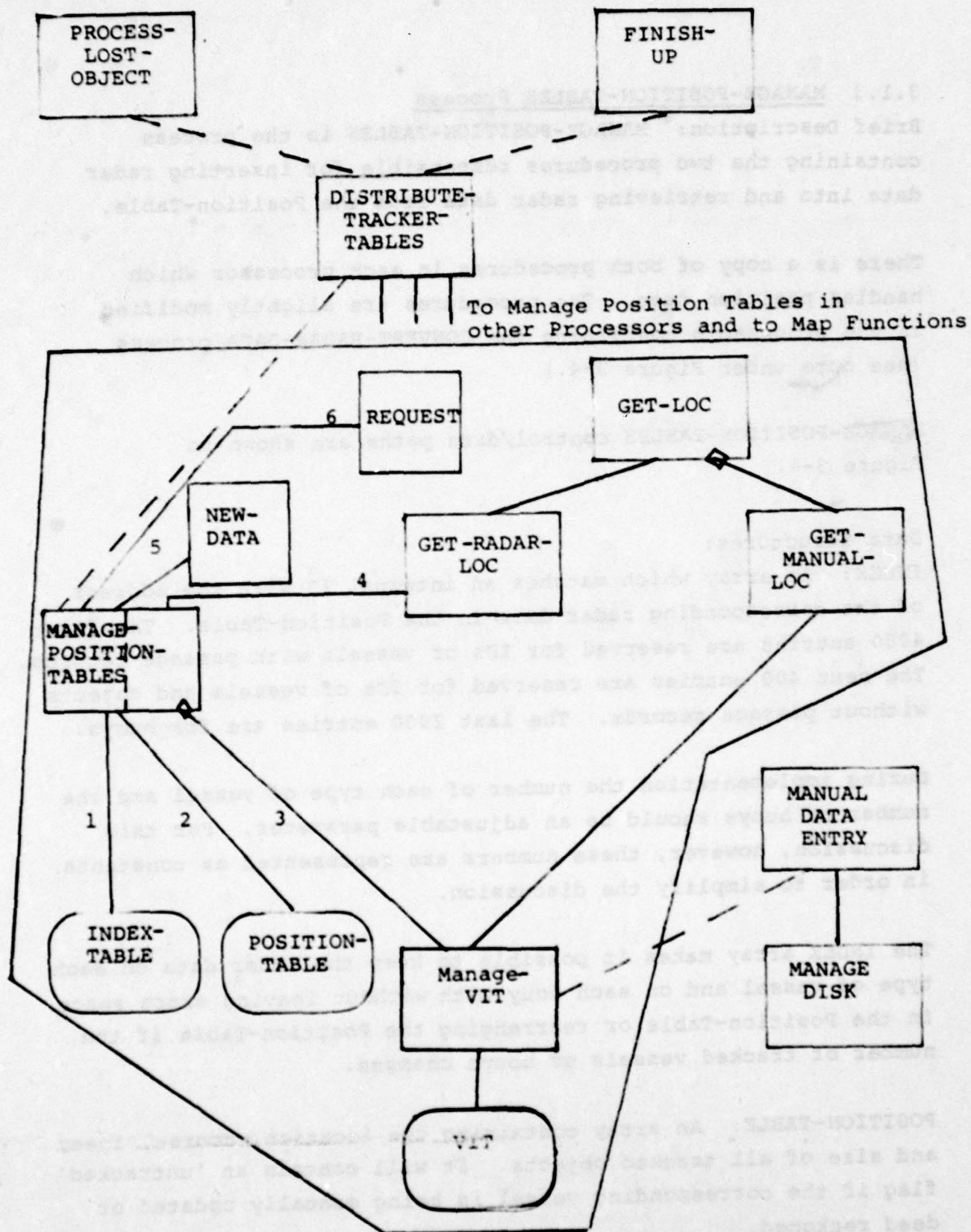


Figure 3-4. MANAGE-POSITION-TABLES

Figure 3-4. MANAGE-POSITION-TABLES

This structure chart shows the interrelations of the MANAGE-POSITION-TABLES (MPT) with other procedures in the case where the MPT is in a different processor than the CONVERT-RADAR-DATA process.

The solid line represents a processor boundary.

- 1 Input: Internal ID  
Output: Address in position table or 'empty'
- 2 Input: Address in position table  
Output: 'Untracked' or position table information
- 3 Input: ID and position table information on vessels which are no longer tracked.  
Output: None
- 4 Input: ID and a list of radar information fields (i.e., loc, course, speed, size)  
Output: The value of each of the fields requested for the object with internal ID = ID.
- 5 Input: New record to be entered in the Position-Table  
Output: None
- 6 Input: Vessel internal ID, list of information requested (i.e., loc, course, speed, size) and the return variable.  
Output: Vessel loc, speed, course and/or size, or any combination of these data fields as specified.

Note:

The internal structure of the MANAGE-POSITION-TABLES (MPT) process is slightly different in the processor containing the CONVERT-RADAR-DATA process. The form of the external requests from procedures needing radar data will be identical. The changes have been made to avoid duplicate radar information in TT & PT within one processor.



# POSITION TABLE

Objects  
tracked  
by  
Radar

ID

Pointer  
12 bits

10 bytes

LOC COURSE SPEED SIZE

0

ID's of  
vessels with  
a passage  
record

max  
4000  
entries

untracked

40K

3999

4000

ID's of  
vessels with  
no passage  
record

7999

8000

ID's of  
Bouys

9999

15K

INDEX

FIGURE 3-5. POSITION TABLE

Process MANAGE-POSITION-TABLES

Procedure NEW-DATA (record):

begin

addr:=INDEX(record.id);

if addr='empty'

then set addr=to next free POSITIONTABLE entry

case type of input

loc, course, speed, size:

POSITION-TABLE(addr):=loc, course, speed, size;

untracked:

do send POSITION-TABLE(addr) and ID to MANAGE-VIT;

POSITION-TABLE(addr):='untracked';

end

empty:

do POSITION-TABLE(addr):='empty';

INDEX(record.id):='empty';

end

end



Procedure REQUEST (id, list, returnvariable);

begin

i:=1;

addr:=index(id);

if addr='empty'

then returnvariable(i):='empty';

else

if positiontable(addr)='untracked'

then returnvariable(i)='untracked';

else

do

if loc in list

then

do returnvariable(i):=positiontable(addr).loc;

i:=i+1; end

if course in list

then

do returnvariable(i):=positiontable(addr).course;

i:=i+1; end

if speed in list

then

do returnvariable(i):=positiontable(addr).speed;

i:=i+1 end

if size in list

then returnvariable(i):=positiontable(addr). size;

end

end

Procedure GET-LOC(ID)

begin

GET-RADAR-LOC (ID);

case returned value

loc: return location to calling process;

untracked:

do GET-MANUAL-LOC(ID)

return loc to calling process;

end;

unused id: return error message to calling process

end

Procedure GET-RADAR-LOC (ID)

begin

find id in index table which matches ID;

Get pointer from index table entry into POSITION-TABLE;

Get loc from POSITION-TABLE entry;

end

Procedure GET-MANUAL-LOC(ID)

begin

Send message to MANAGE-VIT with ID;

Wait for answer containing location from MANAGE-VIT;

end



### 3.2 HAZARD DETECTION PROCESSING

Processes will be included to attempt to detect potentially hazardous situations and provide information (alerts) which will allow the vessel or vessels involved to be notified so the hazard can be avoided.

The VTS Processing/Display Subsystem will have the capability of detecting the following hazards:

- . Potential Collision
- . Lane Stray
- . Route Stray
- . Potential Grounding
- . Excessive Congestion
- . Dangerous Encounters
- . Anchor Drift
- . Navaid Adrift/Missing
- . Excessive Vessel Speed

The task of the software is to recognize these conditions and report the potential hazards to the watchstanders as alerts. In order to recognize these conditions, the software must monitor the data on the location, course and speed of vessels and nav aids.

Recognizing that processing resources will be required to perform these functions, a maximum frequency of execution is given for the monitoring software for each function in the discussion which follows. Each of the required processes will be a cyclic permanent process. The time cycle on which each will operate will be controllable by the Watch Supervisor (see Section 5.4.4, Access System Parameters).

Ideally, the subsystem would generate an alarm only when a truly hazardous condition exists. A variety of normal conditions can, however, be cited which would result in an alarm. To prevent these false alarms the subsystem will also allow the Watch Supervisor to exempt any particular physical area(s) from any or all of these hazard checks. This capability may be used, for example, to prevent false collision warning alerts in narrow channels where vessels pass so close together, even in a normal passing situation, an alert would result.

Additionally, the subsystem will allow watchstanders to exempt particular vessels from any or all of the hazard checks. This capability may be used to avoid false collision alarms for pilot boats, tugs, etc., which have intentional "collisions" as a part of their every day operation, or to prevent lane stray alarms for ferries, etc., which are not normally constrained to lanes.

#### 3.2.1 MONITOR-HAZARD-DETECTION Process

The MONITOR-HAZARD-DETECTION process will control the time schedule for the hazard detection processes. It will base its functions on the time schedule set by the Watch Supervisor and send messages to trigger individual hazard detection processes. When each process completes its cycle, it will send an answer to the monitor process which will record its completion.

If processes fail to complete in the scheduled time, the monitor process will generate error messages indicating that the system has degraded. If the subsystem is operating in a degraded mode, the Watch Supervisor will have the option of adjusting the time schedule or the relative priorities assigned to each type of hazard detection process.

MONITOR-HAZARD-DETECTION control/data paths are shown in Figure 3-6.

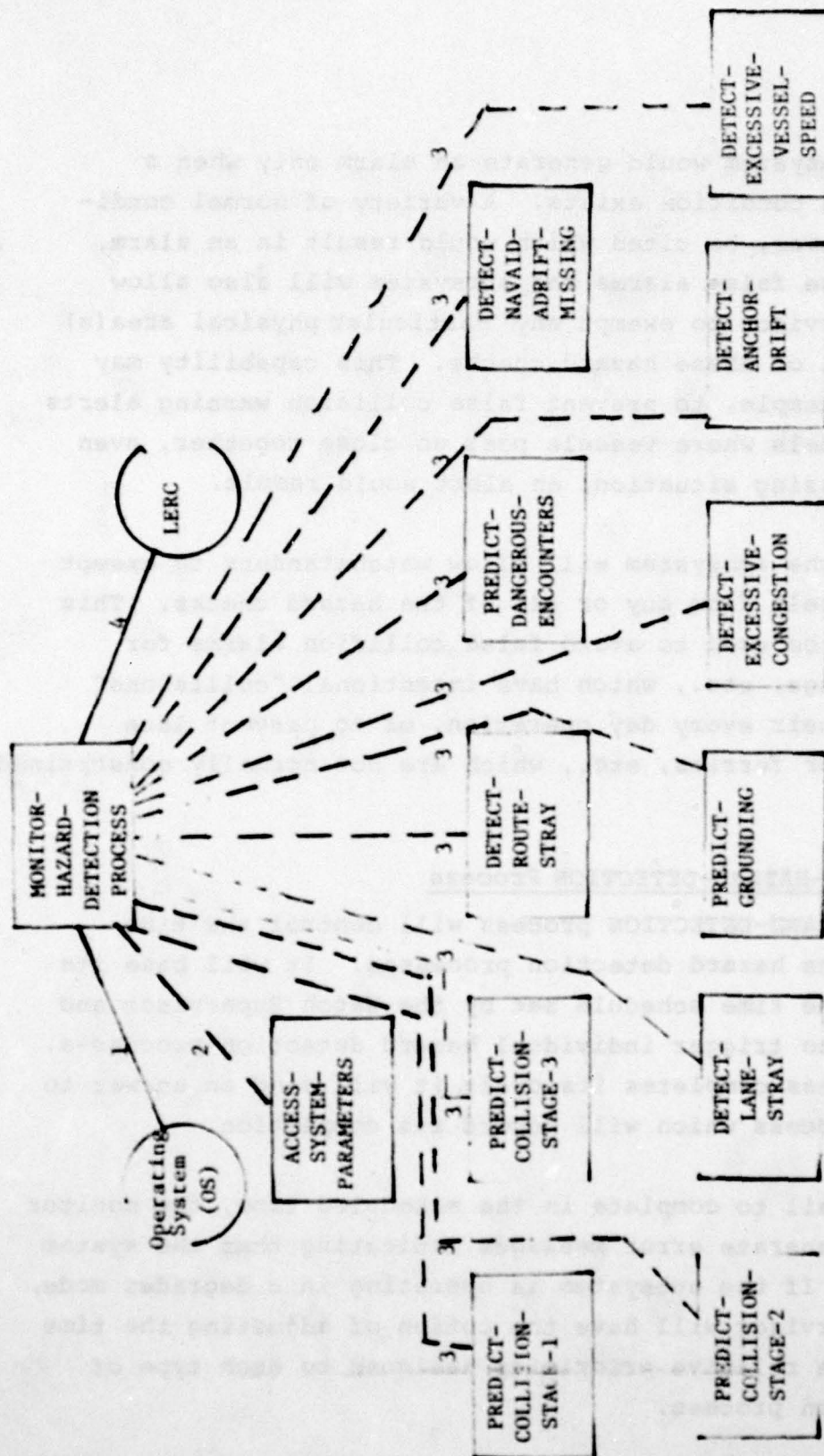


FIGURE 3-6. MONITOR-HAZARD-DETECTION



FIGURE 3-6 MONITOR-HAZARD-DETECTION

- 1     Operating System Service Request: Call the operating  
       requesting that it set a given event flag at a  
       given time of day.  
       Operating System Response: The specified event flag  
       is set at the specified time of day.
- 2     Message: Request for record(s) containing hazard  
       processes cycle times.  
       Answer: Record(s) containing the cycle times of the  
       hazard processes.
- 3     Message: Notify hazard process to begin a new execution  
       cycle.  
       Answer: Notification that the execution cycle has been  
       completed.
- 4     Operating System Service Request: Call the LERC to  
       report that a hazard process is behind schedule  
       Operating System Response: LERC communicates problem  
       to the MERC (if required) and returns control to the  
       MONITOR-HAZARD-DETECTION process.

#### Data Structures:

The major internal data structure of the MONITOR-HAZARD-DETECTION process is the Hazard Process Table. It consists of one entry for each hazard detection process. Each entry consists of the following data items:

- 1) The hazard process name
- 2) The number of the event flag which the operating system is to set when the next execution cycle is to begin
- 3) The Process-In-Execution flag which is set when a message is sent to begin the execution cycle of the hazard detection process. It is cleared when an answer is received indicating that the execution cycle is complete.
- 4) The execution cycle time in minutes and seconds.

Process MONITOR-HAZARD-DETECTION

begin

Send a message to the ACCESS-SYSTEM-PARAMETERS process  
requesting records from the system parameters file containing  
the execution cycle times of the hazard detection processes;

Wait for the answer;

Receive the answer and save the cycle times in the Hazard-  
Process-Table;

for each hazard detection process

do begin

time: = present time + cycle time for hazard detection process;

Issue a delay request to the operating system requesting that  
it set a specified event flag at the above computed time;

Create entry in Hazard-Process-Table containing process  
name, cycle time, and event flag number;

end;

Set the event flag for each hazard detection process;

while true /\*infinite loop\*/

do begin

Determine the lowest number event flag which is set;

Save the event flag number;

Clear the event flag;

case event flag number of

Delay Request:

do begin

Find entry in Hazard-Process-Table with the event  
flag number;

time: = present time + cycle time;

Issue another delay request for the hazard process;

if there is an unanswered message

from the hazard detection process

(Process-In-Execution flag is set);

then

Issue a system service request to the LERC  
indicating that the hazard detection process is  
behind schedule;



```

else
    Send a message to the hazard detection process
    indicating that it should start another execution
    cycle;
    Set the Process-In-Execution flag in the Hazard-
    Process-Table entry;
end;
end;
Answer from a Hazard Process:
do begin
    Clear Process-In-Execution flag for the answering process;
end;
end; /*end case*/
end; /*end begin*/
end; /*end begin*/
end; /*end begin*/

```

### 3.2.2 PREDICT-COLLISION Processes

This section discusses the collision prediction processes and their associated data structures. The following processes perform several different stages of collision prediction: PREDICT-COLLISION-STAGE-1, PREDICT-COLLISION-STAGE-2, and PREDICT-COLLISION-STAGE-3. The data structures are used to provide input information on vessels and to pass data between the collision processes.

PREDICT-COLLISION control/data paths are shown in Figure 3-7.

#### Data Structures:

The PREDICT-COLLISION processes obtain vessel and waterway information by reading the following tables (described in Section 3.1.1): Tracker-Tables, Tracker-ID-Tables, Vessel Information-Table, and Cell-Block-Table.

Tables used internally and used to pass information between the PREDICT-COLLISION processes are described below. The Collision-Table contains an entry for each vessel eligible for collision processing. Each entry contains the vessel internal ID and location (internal coordinates). The Stage-2-List is used to pass vessel pairs from Stage-1 processing to Stage-2 processing. Each entry contains a pair of vessel internal IDs. The Stage-3-List is used to pass vessel pairs from Stage-2 processing to Stage-3 processing, and to keep information on the vessel pair, while it is in Stage-3 processing. Each entry contains 1) a pair of vessel internal IDs, 2) a Potential-Alert-Flag, 3) an Alert-Flag, and 4) a Time-Out field (time of day in minutes, seconds and ticks).

Whenever a process adds an entry to the Stage-2-List or Stage-3-List, it first checks to see if the entry (same vessel pair) is already on the list. The entry can only be added if it is not already on the list. An entry is added by placing it in the first empty (zero-filled) space in the list. To delete an entry from a list, a process zero-fills the entry in the list.

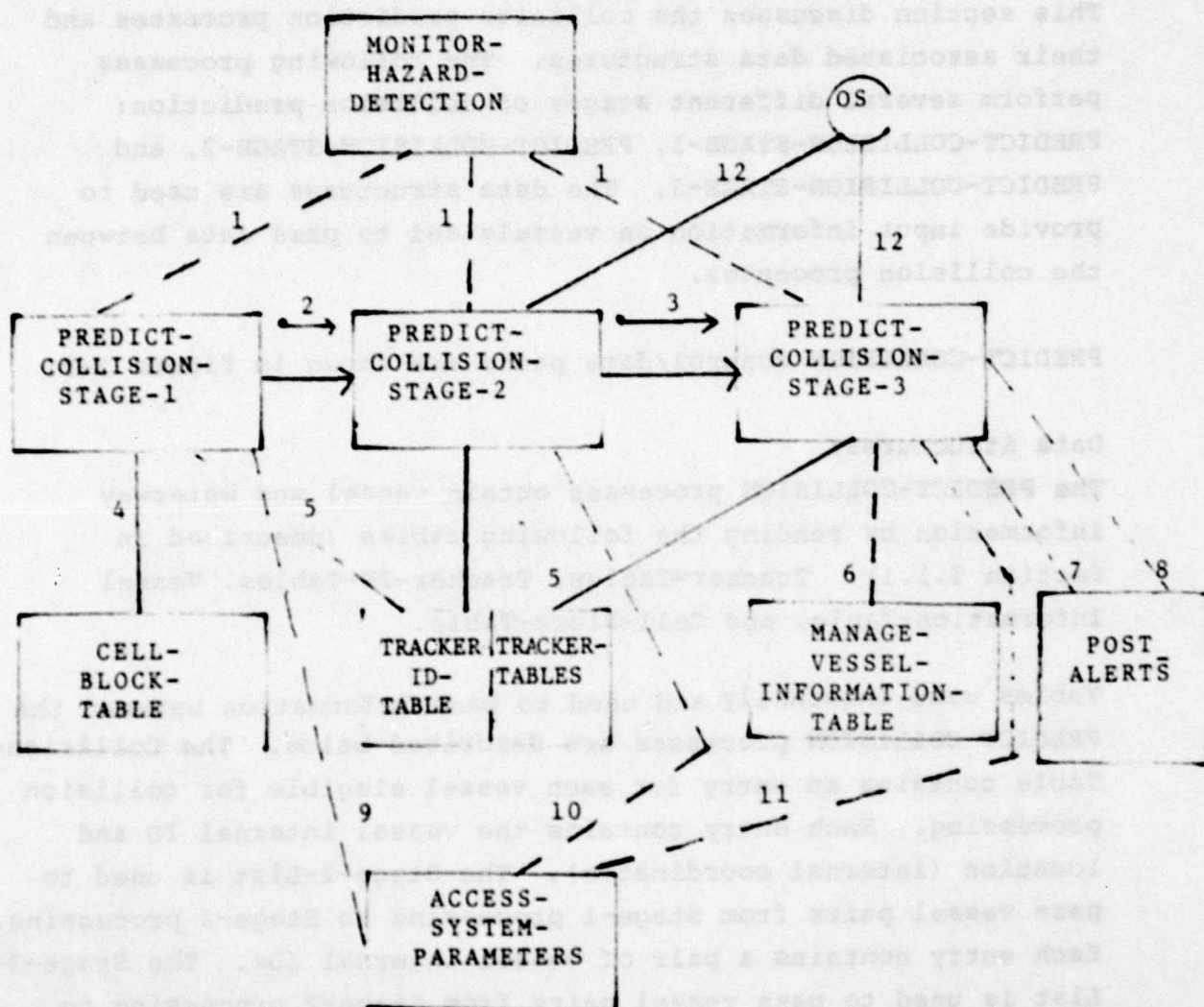


FIGURE 3-7 PREDICT-COLLISION



FIGURE 3- 7 PREDICT-COLLISION

- 1    Message:    Notify collision process to begin a new execution cycle.  
      Answer:    Notify MONITOR-HAZARD-DETECTION process that the execution cycle has been completed.
- 2    Shared Data Area:    Stage-2-List.
- 3    Shared Data Area:    Stage-3-List.
- 4    Global Data Area:    Cell-Block-Table.
- 5    Shared Data Area:    Read-Only access to Tracker-IO-Table and Tracker-Tables.
- 6    Message:    Request for information on vessel of specified internal ID.  
      Answer:    Contents of Vessel-Information-Table entry:
  - 1) Internal ID
  - 2) Vessel External ID code (4-characters)
  - 3) Location (internal coordinates)
  - 4) Course
  - 5) Speed
  - 6) Tracked/Untracked flag
  - 7) Status:    imminent, underway, docked, or anchored
  - 8) Intended Route:    a list of route segment designations
  - 9) Hazard Processing Exemption Flags
- 7    Message:    Notification to enter the alert in the Master-Alert-Queue  
      Answer:    Acknowledgement
- 8    Message:    Notification to delete the alert from the Master-Alert-Queue (cancel the alert)  
      Answer:    Acknowledgement
- 9    Message:    Request Stage-1-Collision-Distance-Threshold  
      Answer:    Stage-1-Collision-Distance-Threshold
- 10    Message:    Request Stage-2-Collision-CPA-Threshold and tCPA-Threshold  
      Answer:    CPA-Threshold and tCPA-Threshold

11 Message: Request vessel size risk component cargo hazard risk code and risk weights  $C_4$  and  $C_5$ , Specified size code for 2 vessels and cargo codes for the vessels.

Answer: Values of requested parameters

12 Operating System Service Request: Time of day

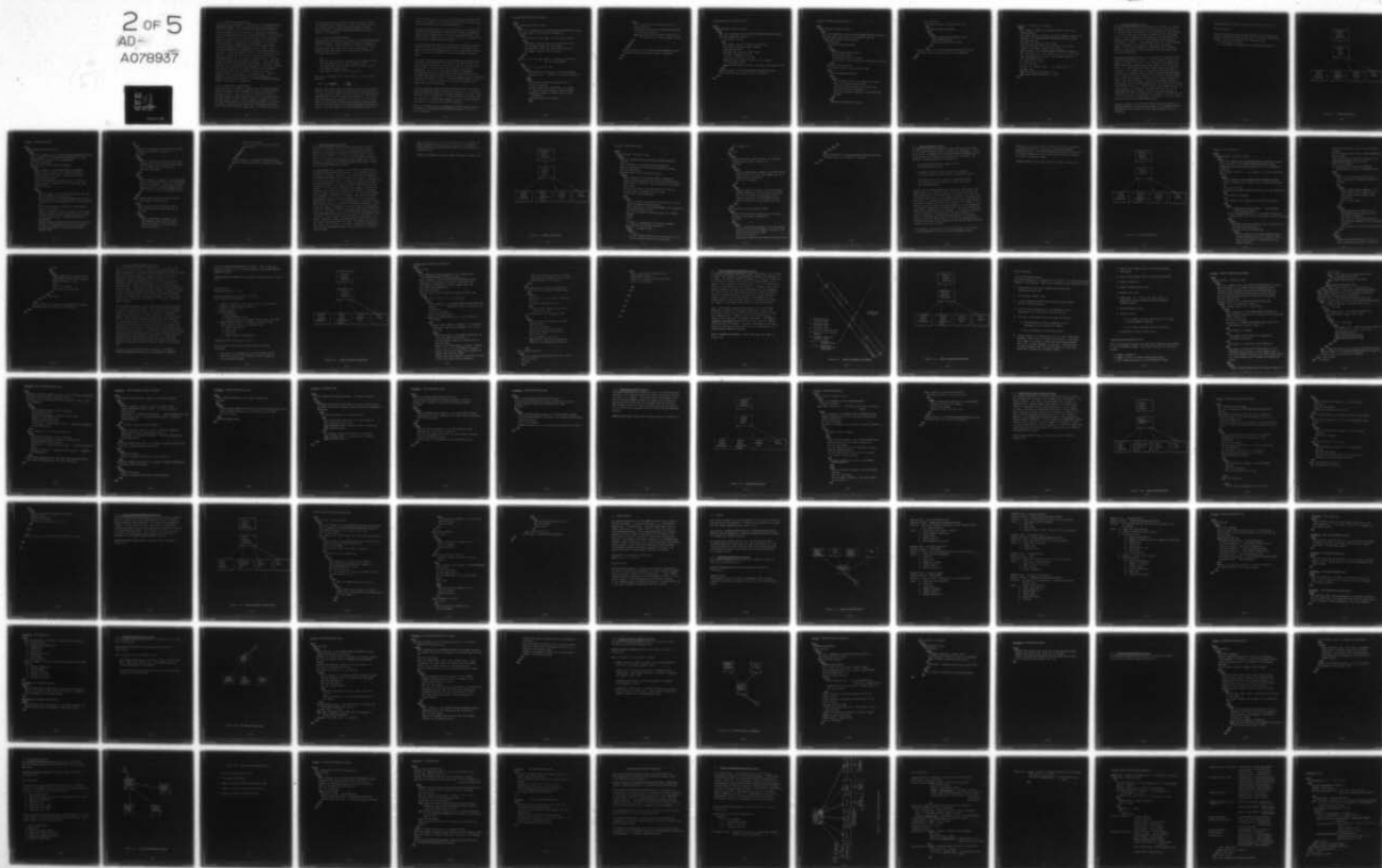
Operating System Response: Current time of day

AD-A078 937

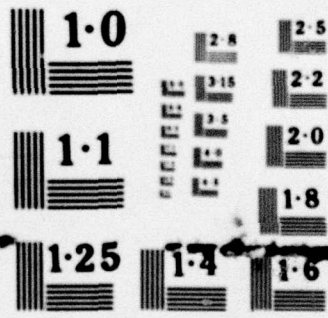
INTERNATIONAL COMPUTING CO BETHESDA MD  
VESSEL TRAFFIC SERVICES PROCESSING/DISPLAY SUBSYSTEM SOFTWARE R--ETC(U)  
SEP 79 C C HENSON , R S GRAHAM , B A MCINTOSH DOT-C6-81-78-1833  
USCG -D-73-79 NL

UNCLASSIFIED

2 OF 5  
AD-  
A078937







NATIONAL BUREAU OF STANDARDS  
MICROCOPY RESOLUTION TEST CHART

#### 3.2.2.1 PREDICT-COLLISION-STAGE-1

This process is activated by a message from the MONITOR-HAZARD-DETECTION process. Such a message is sent at intervals of 30 seconds or longer. Upon receiving this message the STAGE-1 process creates the Collision-Table by extracting information from the Tracker-Tables and the Tracker-ID-Tables. The Collision-Table will contain an entry for each vessel being tracked by radar. Each entry will contain the internal ID of the vessel and its location (internal system coordinates). Once the Collision-Table is completed, an efficiently coded loop will be used to compare the coordinates of every vessel in the Collision-Table with the coordinates of every other vessel in the table. If the difference in the X or Y coordinates is less than the tolerance set by the Watch Supervisor, the Vessel-Information-Table will be checked to determine that 1) neither vessel is exempt from collision processing, 2) both vessels are identified, and 3) both vessels are underway. From each vessel location, the in-core cell block description is checked to ensure that the cell block is not exempt from collision processing. If the vessel pair meets the above criteria, the pair of internal vessel IDs is added to the Stage-2-List. When all vessels in the Collision-Table have been processed, the STAGE-1 process sends an answer to the MONITOR-HAZARD-DETECTION process to notify it of completion.

#### 3.2.2.2 PREDICT-COLLISION-STAGE-2

This process is activated by a message from the MONITOR-HAZARD-DETECTION process. Such a message is sent at intervals of 15 seconds or longer. Upon receiving this message the STAGE-2 process proceeds to check each vessel pair in the Stage-2-List. For each vessel the last location, course, and speed is read from the Tracker Tables (via the Tracker ID Tables). This data is used to calculate the Closest Point of Approach (CPA) and time to CPA (tCPA) from the vessel pair. If both CPA and tCPA

are less than pre-set threshold values, the pair of vessel IDs is added to the Stage-3-List. When all vessel pairs in the Stage-2-List have been checked, the STAGE-2 process sends an answer to the MONITOR-HAZARD-DETECTION process to notify it of completion.

### 3.2.2.3 PREDICT-COLLISION-STAGE-3

This process is activated by a message from the MONITOR-HAZARD-DETECTION process. Such a message is sent at intervals of 6 seconds or longer. Upon receiving this message the STAGE-3 process checks each vessel pair in the Stage-3-List. The processing for each vessel pair includes an estimation of collision risk based on:

- . CPA
- . The rate that the CPA is increasing or decreasing (dCPA)
- . The time remaining until CPA is reached (tCPA)
- . The approximate sizes of the vessels involved (one of four sizes for each vessel)
- . The relative hazard of the cargos aboard.

The risk is estimated using these factors in the following equation:

$$\text{Risk} = C_4 \frac{(S - \text{CPA})}{t\text{CPA}} - C_5 \frac{d\text{CPA}}{t\text{CPA}} + H$$

where  $C_4$  and  $C_5$  are weighting constants for their associated factors. "S" is the vessel size risk factor which has one of 10 values based on a table of risk values for the ten possible combinations of the four vessel size categories. "H" is the risk value based on the hazard of the cargo. Up to 20 cargo types will have values associated with them in the data base. "H" will equal the value of the most hazardous cargo aboard a vessel, or if both vessels are carrying hazardous cargo it is



the sum of the values for the most hazardous cargo aboard each vessel. The tables of size and cargo risk values are data base constants accessible by the Watch Supervisor.

The data for performing the above calculations is obtained from several different sources. The location, course, and speed is read from the Tracker-Tables (via the Tracker-ID-Tables). The vessel status, exemption flags, cargo type and size category are read from the Vessel-Information-Table.

If the computed risk value for a vessel pair exceeds the preset threshold value, the Potential-Alert-Flag in the Stage-3-List entry is set and the Time-Out field in the Stage-3-List entry is set to the time of day when the risk value should be checked again.

After all Stage-3-List entries are processed as described above, the STAGE-3 process checks each entry in the Stage-3-List. If the Potential-Alert-Flag is set and the Time-Out field is less than or equal to the current time of day, the risk value is recomputed. If the risk value now exceeds the threshold, the Alert-Flag is set (in the entry), the Potential-Alert-Flag is reset, the Time-Out field is set to the time of day for rechecking, and an alert message is sent to the POST-ALERTS process. On the other hand, if the risk value is less than the threshold, the Stage-3-List entry is deleted.

After all potential alert entries have been checked, the STAGE-3 process checks each entry with the Alert-Flag set. For any entry where the Time-Out field is less than or equal to the time of day, the risk value is recomputed. If the risk value is now less than the threshold, a Cancel-Alert message is sent to the POST-ALERT process, and the Stage-3-List entry is deleted.

When the processing cycle is complete the STAGE-3 process sends an answer to the MONITOR-HAZARD-DETECTION process to notify it of the completion.

Process PREDICT-COLLISION-STAGE-1

begin

while true do /\*infinite loop\*/

begin

Wait for a message from the MONITOR-HAZARD-DETECTION process;

for each entry in the Tracker-ID-Tables do

begin

if Internal ID <4000 then /\*if identified vessel\*/

begin

Get vessel location from the Tracker-Table entry  
of the radar number and track number;

Add an entry to the Collision-Table containing  
the vessel internal ID and location;

end;

end;

for i: = 1 to (n-1) do /\*where n = number of entries in  
the Collision-Table\*/

begin

for j: = (i + 1) to (n-1) do

begin

if difference in latitude or longitude between  
entries i and j < threshold (a system parameter)

then

begin

Check Vessel-Information-Table entries  
for the 2 vessels;

if Both vessels are identified (i.e., have a  
vessel ID code (external 4-letter code))

and Neither vessel is exempt from collision  
processing

and Both vessels are underway

then



```

begin
  Get in-core cell block descriptions for
  both positions;
  if Neither cell block is flagged exempt from
    collision processing
  then Add the pair of internal vessel ID's to
    the Stage-2-List (if not already in list);
  end;
end;
end;
end;
  Send an answer to the MONITOR-HAZARD-DETECTION process
  indicating that the execution cycle is complete;
end;
end

```



Process PREDICT-COLLISION-STAGE-2

begin

while true do /\*infinite loop\*/

begin

Wait for a message from the MONITOR-HAZARD-DETECTION process;

for each vessel pair entry in

the Stage-2-List do

begin

Get vessel location, course, and speed for  
both vessels from the Tracker-Table;

Calculate the CPA;

Calculate the time until CPA (tCPA);

if (CPA < threshold for CPA

or tCPA < threshold for tCPA)

and the vessel internal ID pair is not already  
on the Stage-3-List

then Add the internal ID pair with CPA to the Stage-3 List'

end;

Send an answer to the MONITOR-HAZARD-DETECTION process  
indicating that the execution cycle is complete;

end;

end;

Process PREDICT-COLLISION-STAGE-3

begin

while true do /\*infinite loop\*/

begin

Wait for a message from the MONITOR-HAZARD-DETECTION process;

for each vessel pair entry in the STAGE-3-LIST do

case STAGE-3-LIST entry flags of

No-Flags-Set:

if Risk-Threshold-Exceeded

then

begin

Set Potential-Alert-Flag in

Stage-3-List entry;

Set Time-Out field in Stage-3

-List entry to time of day for rechecking this entry;

end;

Potential-Alert-Flag Set:

if Time-Out field  $\leq$  Current Time of Day

then

if Risk-Threshold-Exceeded

then

begin

Set the Alert Flag in the Stage-3-List entry;

Clear the Potential-Alert-Flag;

Set the Time-Out field to the time of day

the entry is to be rechecked;

Send an Enter-Alert message to the POST-ALERTS  
process

end;

else

Delete the Stage-3-List entry;



Alert-Flag Set:

if Time-Out field  $\leq$  Current Time of Day

then

if Risk-Threshold-Exceeded

then

begin

Send a Cancel-Alert message to the

POST-ALERTS process;

Delete the Stage-3-List entry

end;

end;

Send an answer to the MONITOR-HAZARD-DETECTION process  
indicating that the execution cycle is completed;

end;

end;



Function RISK-THRESHOLD-EXCEEDED

begin

Get location, course, and speed for both Vessels from  
Tracker-Tables;

Get vessel sizes,  $S_1$  and  $S_2$ , and cargo codes,  $C_1$  and  $C_2$ , by  
message to the MANAGE-VESSEL-INFORMATION-TABLE process;

Calculate the CPA;

Calculate the time until CPA (tCPA);

Get previous CPA (PCPA) from the Stage-3-List entry;

dCPAU: = (CPA-pCPA)/(Stage 2 cycle time);

Send message to the ACCESS-SYSTEM-PARAMETERS process to  
get the vessel size risk components  $R_s$  ( $S_1$ ,  $S_2$ ) and cargo  
risk components  $A(C_1)$ ,  $A(C_2)$ , and weights  $C_4$  and  $C_5$ ;  
and risk-threshold,  $T_R$ ;

$H$ : = max ( $H(C_1)$ ,  $H(C_2)$ );

$S$ : =  $R_s$  ( $S_1$ ,  $S_2$ );

Risk: =  $\{C_u (S-CPA)/(tCPA)\} - \{C_s (dCPA)/tCPA\} + H$ ;

if Risk  $\geq T_R$

then Risk-Threshold-Exceeded: = true

else Risk-Threshold-Exceeded: = false;

end;

### 3.2.3 DETECT-LANE-STRAY Process

The DETECT-LANE-STRAY process determines if a vessel is outside the proper lane of travel or will be outside the lane within 'n' minutes, based on straight line dead reckoning. The constant 'n' will be a data base constant accessible by the Watch Supervisor. A lane is defined in the system as the area within the minimum width distance of the route segment currently being traversed by a vessel. If the minimum width of the route segment is not specified, a lane does not exist along the route segment. Lane stray processing is performed only on vessels which are identified, in-track, underway, and not exempt from lane stray processing.

The DETECT-LANE-STRAY process is activated by a message from the MONITOR-HAZARD-DETECTION process. Such a message is sent at 30 second (or longer) intervals. Upon receiving this message the DETECT-LANE-STRAY process checks each vessel in the Collision-Table (i.e., each identified vessel being tracked by level 4 or 5 sensors) with information in the Vessel-Information-Table to see that it is underway and not exempt from lane stray processing. If the vessel meets these criteria, its location, course, speed and current route segment are read from the Vessel-Information-Table. The route subsegment (straight line segment) endpoint coordinates and the minimum segment width are read from an in-core Route-Segment-Table or, if the particular route segment data is not in the table, it is read into the table from the disc via the ACCESS-WATERWAY-DATA process. Based on the data gathered, a computation is made to determine if the vessel is outside the lane or will be outside the lane within 'n' minutes. If so, an alert message is sent to the POST-ALERT process.

When all vessels in the Collision-Table have been checked, the DETECT-LANE-STRAY process sends an answer to the MONITOR-HAZARD-DETECTION process notifying it that the processing cycle is completed.

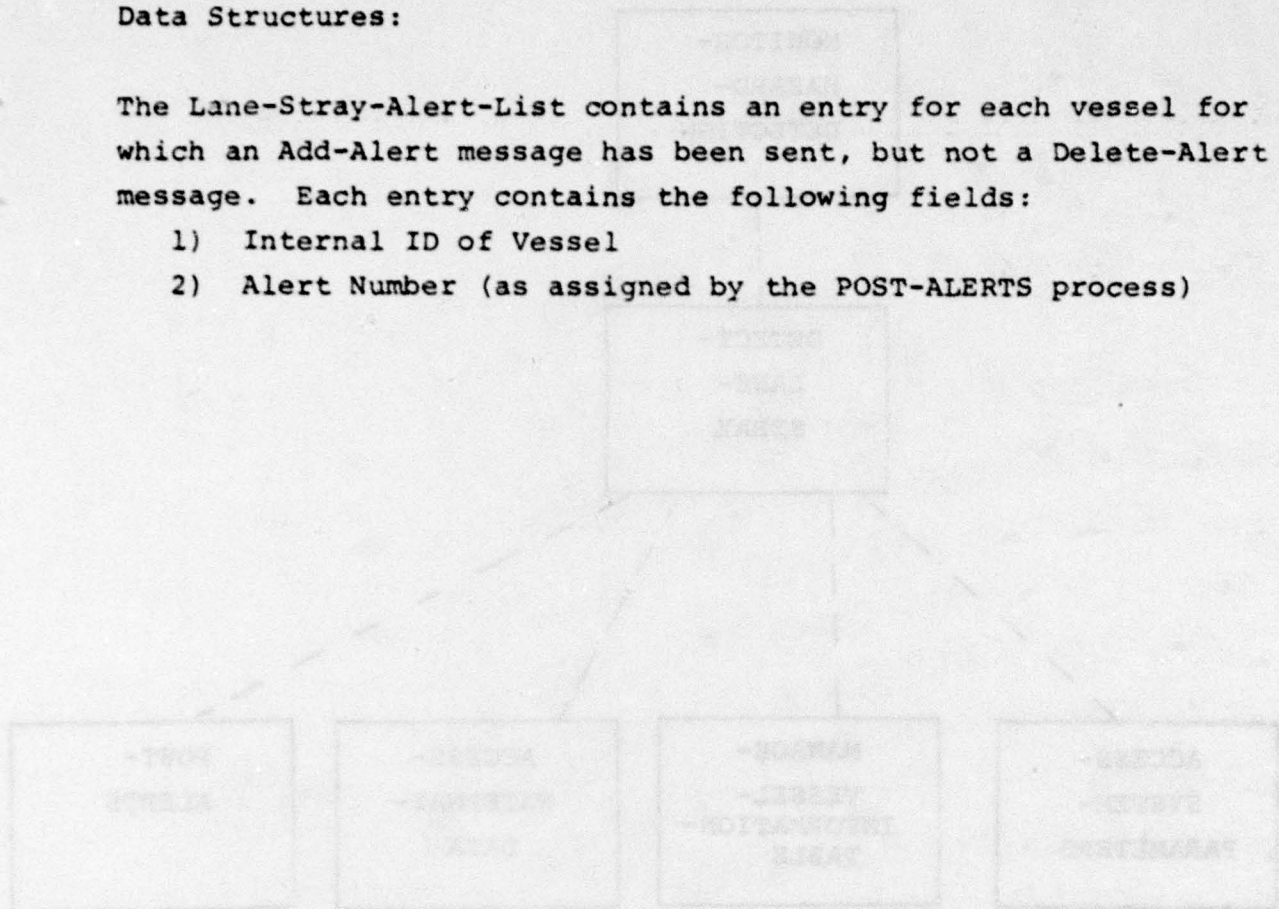


DETECT-LANE-STRAY control/data paths are shown in Figure 3-8.

Data Structures:

The Lane-Stray-Alert-List contains an entry for each vessel for which an Add-Alert message has been sent, but not a Delete-Alert message. Each entry contains the following fields:

- 1) Internal ID of Vessel
- 2) Alert Number (as assigned by the POST-ALERTS process)





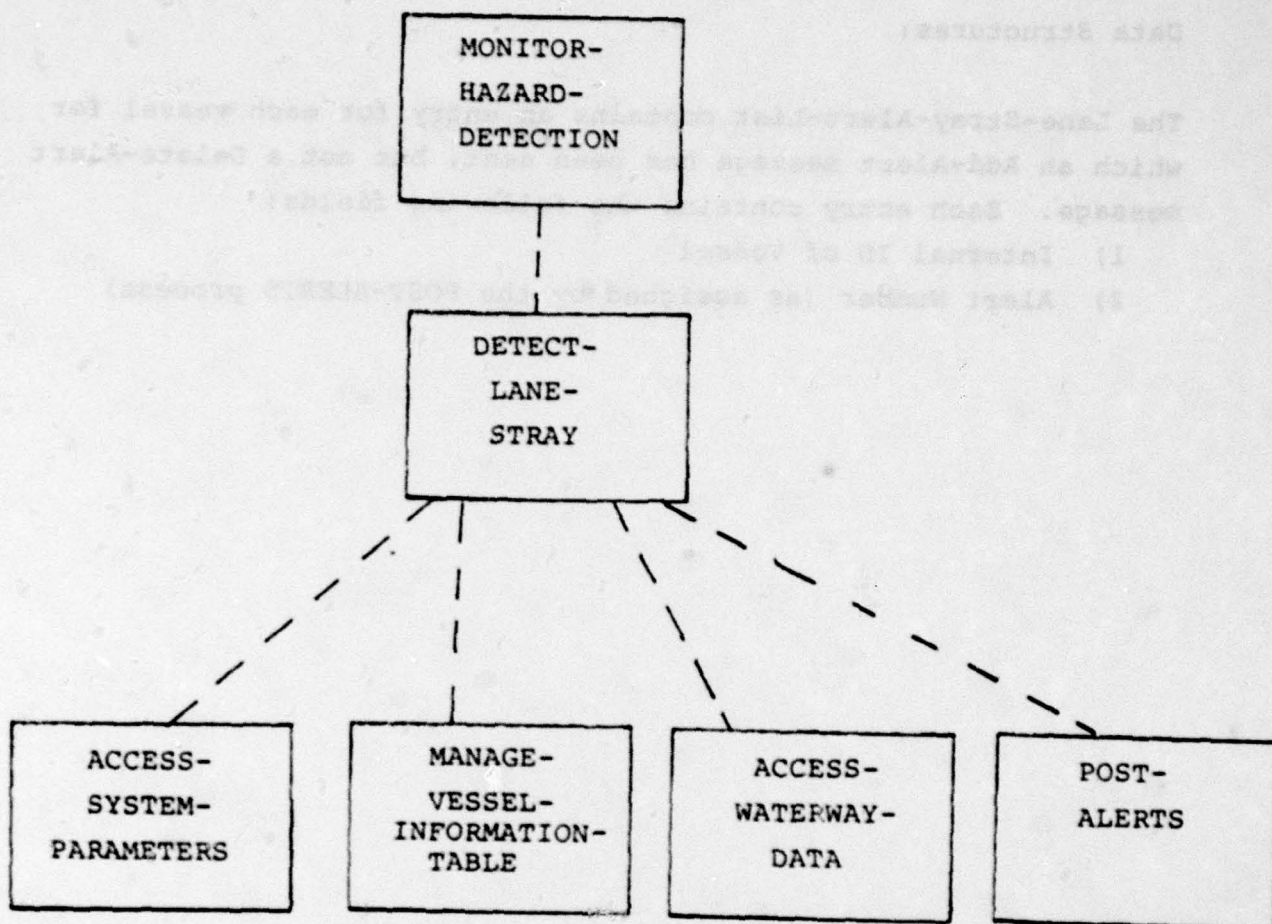


Figure 3-2 DETECT-LANE-STRAY

Process DETECT-LANE-STRAY

begin

while true do /\*infinite loop\*/

begin

Wait for a message from the MONITOR-HAZARD-DETECTION process;

Send a message to the ACCESS-SYSTEM-PARAMETERS process  
requesting the Lane-Stray-Look-Ahead-Time;

Wait for the answer;

for each entry in the Collision-Table do

begin

Send a message to the MANAGE-VESSEL-INFORMATION-  
TABLE process requesting the Vessel-Information-  
Table record of the vessel internal ID from the  
Collision-Table entry;

Wait for the answer (containing the V-I-T record);

if the vessel is underway and not exempt from lane  
stray processing

then

begin

Get the current route segment designation from the  
Vessel-Information-Table record;

Send a message to the ACCESS-WATERWAY-DATA process  
requesting the route segment record of the specified  
ID;

Wait for the answer (contains the route segment  
record if it exists);

Based on location, course, and speed in the Vessel-  
Information-Table record, dead-reckon the position  
at the (current time) + (the Lane-Stray-Look-  
Ahead-Time)'

Compute distance of dead-reckoned position from the  
closest route subsegment (straight line segment);

if computed distance  $\geq$  minimum route segment  
channel width



then

begin

Search the Lane-Stray-Alert-List for an entry  
containing the vessel internal ID;

if found

then

begin

Send an Update-Alert message to the POST-  
ALERTS process containing the Alert Number  
and the update alert information;  
Wait for the answer;

end;

else

begin

Send an Add-Alert message to the POST-ALERTS  
process containing the new alert information;  
Wait for the answer (containing the Alert-  
Number assigned by the POST-ALERTS process;  
Add an entry to the Lane-Stray-Alert-List

end;

end;

else /\*dead-reckoned distance within minimum  
route segment channel width\*/

begin

Search the Lane-Stray-Alert-List for an  
entry containing the vessel internal ID;

if found

then

begin

Send a Delete-Alert message to the  
POST-ALERTS process with the Alert-  
Number contained in the Lane-Stray-  
Alert-List entry;



Wait for an answer;

Delete the entry from the Lane-Stray-  
Alert-List;

end;

end;

end;

end;

Send an answer to the MONITOR-HAZARD-DETECTION  
process indicating that the iteration is complete;

end;

end;

#### 3.2.4 DETECT-ROUTE-STRAY Process

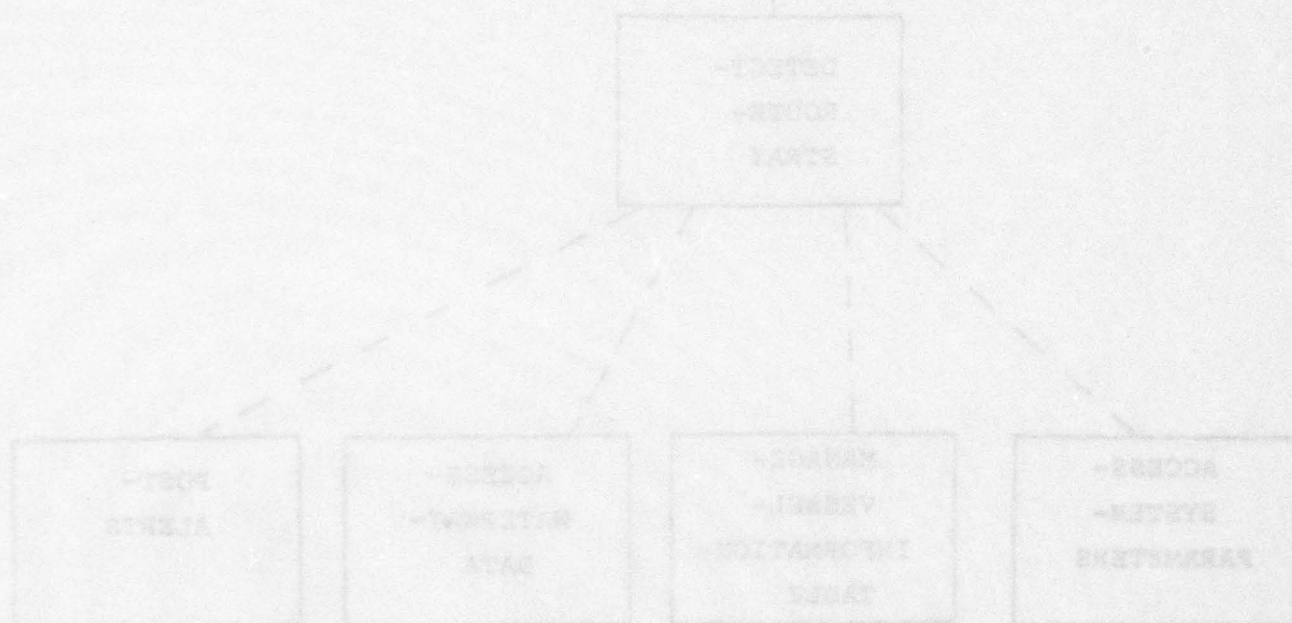
The DETECT-ROUTE-STRAY process detects the condition where a vessel is in track by Level 4 or 5 sensors and the tracked position is not within a predefined threshold distance ( $C_{rs}$ ) from any route segment of a vessel's intended route. When a vessel is in a Level 1, 2, or 3 area and the reported waypoint does not lie upon a route segment of the intended route, this condition is handled by the ACCESS-PASSAGE-FILE process, rather than the DETECT-ROUTE-STRAY process.

The DETECT-ROUTE-STRAY process is activated by a message from the MONITOR-HAZARD-DETECTION process. Such a message is sent at 30 second (or longer) intervals. Upon receiving this message, the DETECT-ROUTE-STRAY process obtains each vessel ID and location in the Collision-Table. For each such vessel it performs the following processing. First the Vessel-Information-Table entry is checked to see if the vessel is exempt from route stray processing. If not, then the intended route segment list and the index into the list of the current route segment are obtained from the Vessel-Information-Table entry. For this route segment, the subsegments (straight line segments) are obtained from the Route-Segment-Table or the Route Segment File (via the ACCESS-WATERWAY-DATA process). For each subsegment, the process determines whether the line is within the threshold distance. If none of the straight line segments are close enough, the Route-Stray-Alert-List (RSAL) is checked to see if an alert has already been issued for the vessel. If so, an Update-Alert message is sent to the POST-ALERTS process with the last location and distance from the route. If the vessel ID is not already in the RSAL, it is added. Whenever a vessel is checked and is found to be within the threshold distance from the route, the process searches for the vessel ID in the RSAL. If found, a Cancel-Alert message is sent to the POST-ALERTS process, and the vessel ID is deleted from the RSAL.



When all vessels in the Collision-Table have been checked, the DETECT-ROUTE-STRAY process sends an answer to the MONITOR-HAZARD-DETECTION process indicating that it has completed the processing cycle.

DETECT-ROUTE-STRAY control/data paths are shown in Figure 3-9.





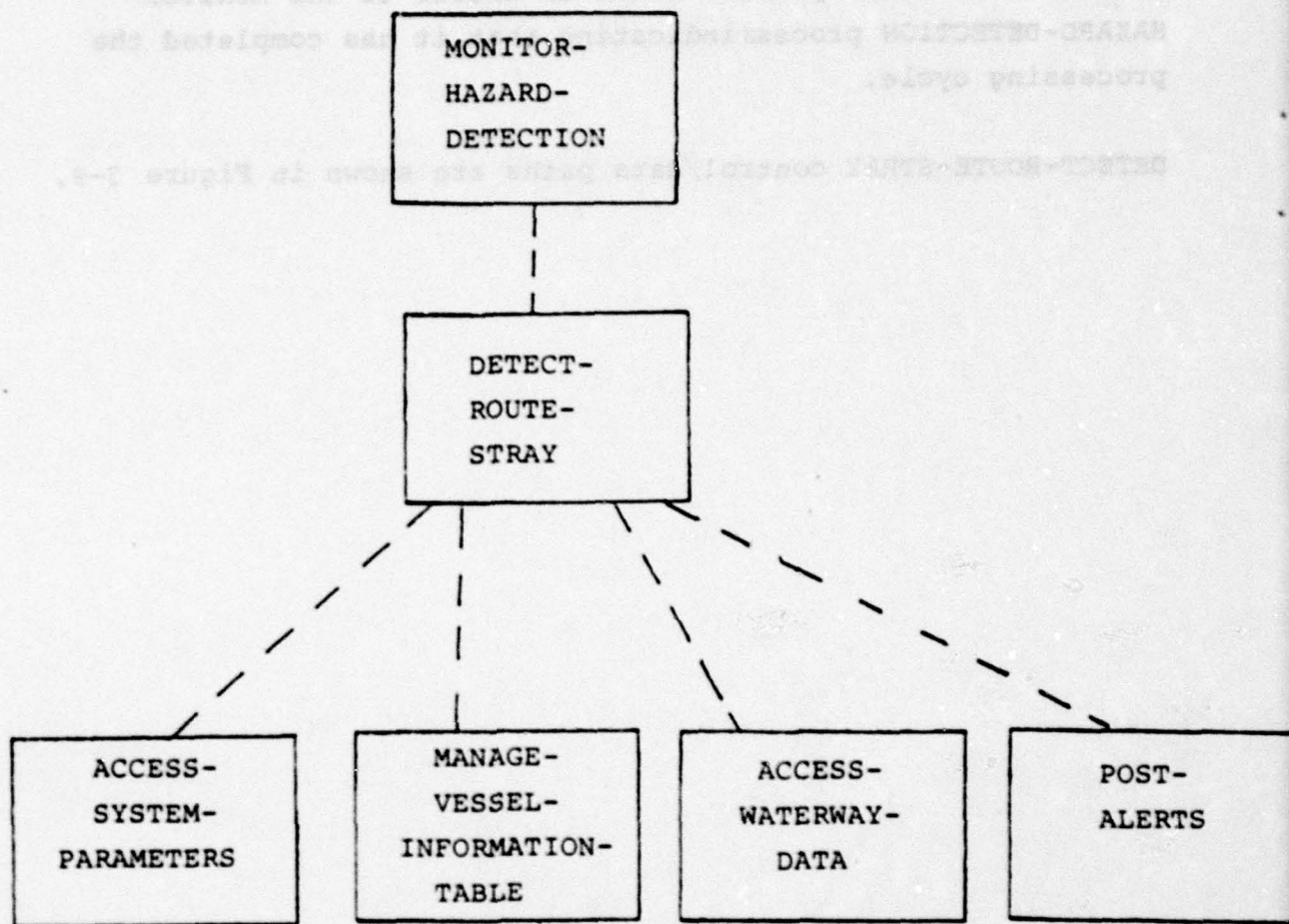


Figure 3-9 DETECT-ROUTE-STRAY

Process DETECT-ROUTE-STRAY

begin

while true do /\*infinite loop\*/

begin

Wait for a message from the MONITOR-HAZARD-DETECTION  
process;

Send a message to the ACCESS-SYSTEM-PARAMETERS  
process requesting the Route-Stray-Threshold-Distance;  
Wait for the answer;

for each entry in the Collision-Table do

begin

Send a message to the MANAGE-VESSEL-INFORMATION-  
TABLE process requesting the Vessel-Information-Table  
record with the internal vessel ID from the Collision-  
Table entry;

Wait for the answer (containing the V-I-T record);

if the vessel is underway and not exempt  
from route stray processing

then

begin

Get the current route segment designation from the  
Vessel-Information-Table record;

Send a message to the ACCESS-WATERWAY-DATA requesting  
the route segment record;

Wait for the answer (containing the route segment  
record);

D:=00;

i:=0

n:=number of subsegments in the route segment;

C<sub>rs</sub>:=Route-Stray-Threshold-Distance;

While D > C<sub>rs</sub> and i < n do

begin

i:=i+1;

d:=the minimum distance from the vessel position  
to the closest point on the subsegment;



```

    if d D then D:=d;
  end;
if D Crs
then
  begin
    Search the Route-Stray-Alert-List for an entry
    containing the vessel internal ID;
    if found
    then
      begin
        Send an Update-Alert message to the POST-ALERTS
        process containing the Alert Number and the
        update alert information;
        Wait for the answer;
      end
    else
      begin
        Send an Add-Alert message to the POST-ALERTS
        process containing the new alert information;
        Wait for the answer containing the Alert-
        Number assigned by the POST-ALERT process;
        Add on entry to the Route-Stray-Alert-List
      end;
    end;
  else /*vessel is within channel width*/
    begin
      Search the Route-Stray-Alert-List for an entry
      containing the vessel internal ID;
      if found
      then
        begin
          Send a Delete-Alert message to the POST-ALERTS
          process with the Alert-Number contained in the
          Route-Stray-Alert-List entry;
          Wait for an answer;
          Delete the entry from the Route-Stray-Alert-List;
        end
      end
    end
  end

```



end;  
end;  
end;  
end;  
Send an answer to the MONITOR-HAZARD-DETECTION process  
indicating that the iteration is complete;  
end;  
end;

### 3.2.5 PREDICT-GROUNDING Process

The PREDICT-GROUNDING process will check each identified vessel in the Vessel-Information-Table for potential grounding. It will be activated by messages from the MONITOR-HAZARD-DETECTION process which are sent at intervals not shorter than 30 seconds. The potential grounding check will be based on:

- . The reported draft of the vessel from the Vessel-Information-Table
- . The depth of water at MLLW in the route segments or cells through which the vessel is expected to pass.
- . The present water level relative to MLLW from water level sensors, tide schedules or manual entry from the Current-Tide-File.

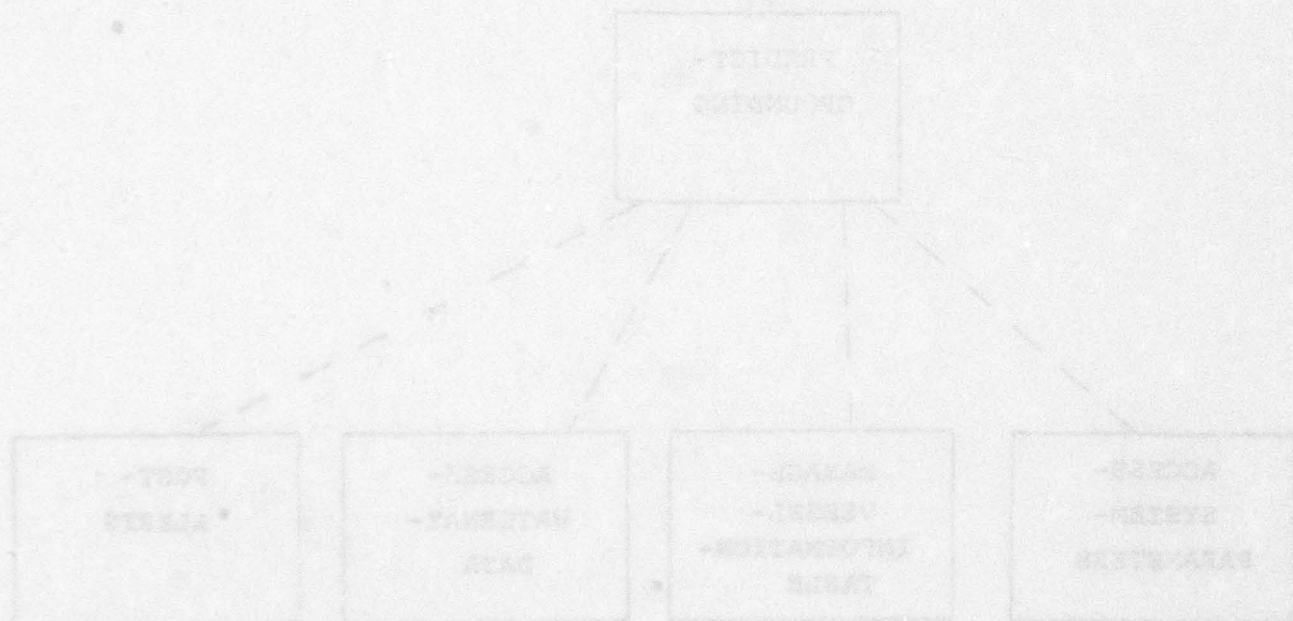
The potential grounding detection process will dead reckon each vessel ahead up to a preset amount of time (as selected by the Watch Supervisor but not longer than 10 minutes) and verify that the actual water level on its intended route over the preset time period (i.e., the depth at MLLW plus the height of water above or below MLLW) exceeds the draft of the vessel. If a vessel is following a route structure, this dead reckoning will be based upon its reported speed and its intended route, and will consider the depth of the appropriate route segments. If, however, the vessel is not following a route structure, the dead reckoning will revert to straight line dead reckoning based on the vessel's current course and speed, and will use for the grounding determinations the depth of that portion of the waterway which the dead reckoned track will traverse.

If grounding is projected on the current course of the vessel, an alert message will be sent to the POST-ALERTS process.



When each cycle of the grounding detection process is completed a completion message will be sent to the MONITOR-HAZARD-DETECTION process. The PREDICT-GROUNDING process will then wait for another message from the monitor process before repeating the cycle of grounding checks.

PREDICT-GROUNDING control/data paths are shown in Figure 3-10.





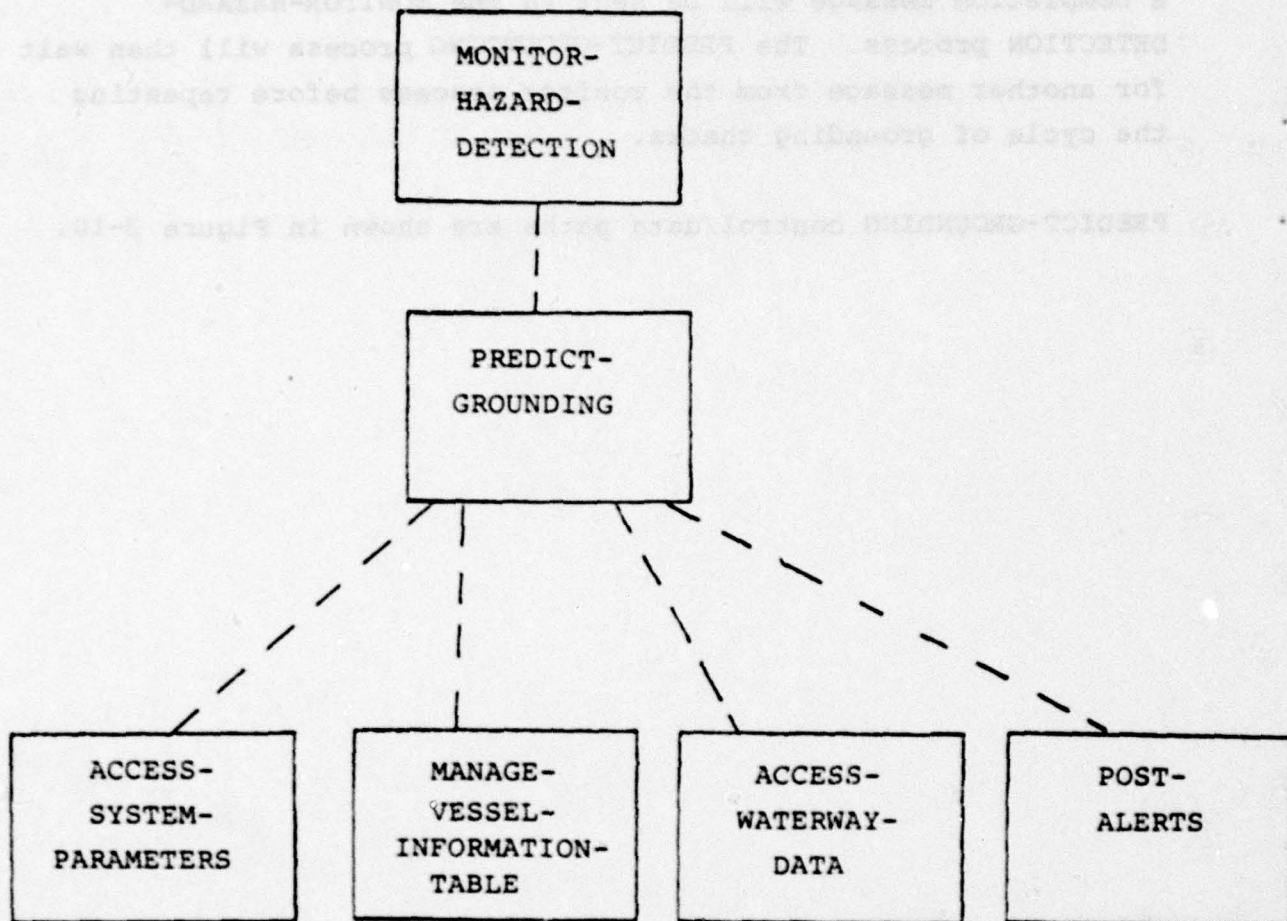


Figure 3-10 PREDICT-GROUNDING

Process PREDICT-GROUNDING

begin

while true do /\*infinite loop\*/

begin

Wait for a message from the MONITOR-HAZARD-DETECTION process;  
Send a message to the ACCESS-SYSTEM-PARAMETERS process  
requesting the Grounding-Look-Ahead-Time and the Grounding-  
Threshold.

for internal vessel ID: = 1 to 3999 do /\* for each identified  
vessel\*/

begin

Send a message to the MANAGE-VESSEL-INFORMATION-TABLE  
process requesting the record with the specified internal  
ID;

Wait for an answer;

if there is a Vessel-Information-Table with the internal ID  
and

the vessel is underway

and

the vessel is not exempt from grounding processing

then

begin

D: = (Grounding-Look-Ahead-Time) \* (vessel speed);  
/\*Look Ahead Distance\*/

if the Intended-Route-List is the Vessel-Information  
Table record is not empty /\*following a route structure\*

then

begin

Determine the route segments which the vessel will  
traverse within distance D;

for each such route segment do

begin

Send a message to the ACCESS-WATERWAY-DATA  
process requesting the route segment record;  
Wait for the answer (containing the route  
segment record);



```

Get the Minimum-Depth along the route segment
at MLLW;
Get the Vessel-Draft from the Vessel-Information-
Table record;
Send a Get-Current-Tide-Record message to the
ACCESS-ENVIRONMENTAL-DATA process;
Wait for the answer;
if (Vessel-Draft)+(Grounding-Threshold)
    (Minimum-Depth)+(Level of Tide Above MLLW)
then
    begin
        Search the Grounding-Alert-List for an
        entry containing the vessel internal ID;
        if found
            then
                begin
                    Send an Update-Alert message to the
                    POST-ALERTS process containing the
                    Alert Number and the update alert
                    information;
                    Wait for the answer;
                end;
            else
                begin
                    Send an Add-Alert message to the
                    POST-ALERTS process containing the
                    new alert information;
                    Wait for the answer containing the
                    Alert-Number assigned by the POST-ALERTS
                    process;
                    Add on entry to the Grounding-Alert-List
                end;
            end;
        else /*
            begin
                Search the Grounding-Alert-List for an
                entry containing the vessel internal ID;

```



```

    if found
    then
        begin
            Send a Delete-Alert message to the
            POST-ALERTS process with the Alert-
            Number contained in the
            entry
            Wait for an answer;
            Delete the entry from the
        end;
    end;
end; /*if D>Crs*/
end;
end;
end;
end;
end;
end;

```

Send an answer to the MONITOR-HAZARD-DETECTION process indicating that the iteration is complete;

### 3.2.6 DETECT-EXCESSIVE-CONGESTION Process

The DETECT-EXCESSIVE-CONGESTION process will detect when more vessels are in or will be in a critical area than the area can safely accommodate. The checks will be repeated at a rate not faster than once every 30 seconds. Up to 40 critical areas will be defined by a center point and a radius distance (in a two dimensional harbor representation) or by waypoints and linear distances along routes from waypoints (in a one dimensional representation). Each critical area will be assigned a maximum capacity value. The amount of actual congestion will be determined using the technique defined below which yields a congestion value. If this value will exceed the critical area's maximum capacity within a pre-established look-ahead time span (not greater than 30 minutes), a congestion alert message will be sent to the POST-ALERTS process.

The look-ahead time span will be a data base constant accessible by the Watch Supervisor station. The look-ahead will be accomplished by advancing all vessels which are in a route structure along their intended route at their reported (or measured) speed. Vessels outside a route structure will not be considered. The value each vessel contributes toward this total capacity will be dependent upon its size and cargo risk value. A table of capacity contribution constants for each of four vessel size categories will be stored in the data base and be accessible by the Watch Supervisor station. The capacity contribution resulting from the various hazardous cargos will be the same as the 20 values established for these cargoes in the potential collision hazard detection process described in subsection 3.2.2. The actual congestion of an area will be the sum of the size and cargo capacity contribution constants from each vessel in the critical area.

As with the other hazard detection processes, the DETECT-EXCESSIVE-CONGESTION process will be initiated by a message



from the MONITOR-HAZARD-DETECTION process. When it has completed its checks, it will send an answer to the MONITOR-HAZARD-DETECTION process.

DETECT-EXCESSIVE-CONGESTION control/data paths are shown in Figure 3-11.

#### Data Structure:

##### Congestion-Area-List (C-A-L)

For each congestion area there is an entry which contains the following;

- 1) Waypoint designation and coordinate of the area center;
- 2) Maximum distance from the waypoint
- 3) Maximum capacity value
- 4) Route segments
  - a) designation
  - b) Inbound/Outbound flag
  - c) Distance along route segment where enters/exits area
- 5) Vessel-Congestion-List - contains an entry for each vessel passing through the congestion area
  - a) Vessel internal ID
  - b) Entry time
  - c) Exit time
  - d) Vessel capacity contribution

##### Congestion-Alert-Issued-List (C-A-I-L)

For each alert which has been issued there is an entry containing:

- 1) The alert-ID (as assigned by the POST-ALERTS process)
- 2) Index into the C-A-L (to specify to which congestion area the alert applies)



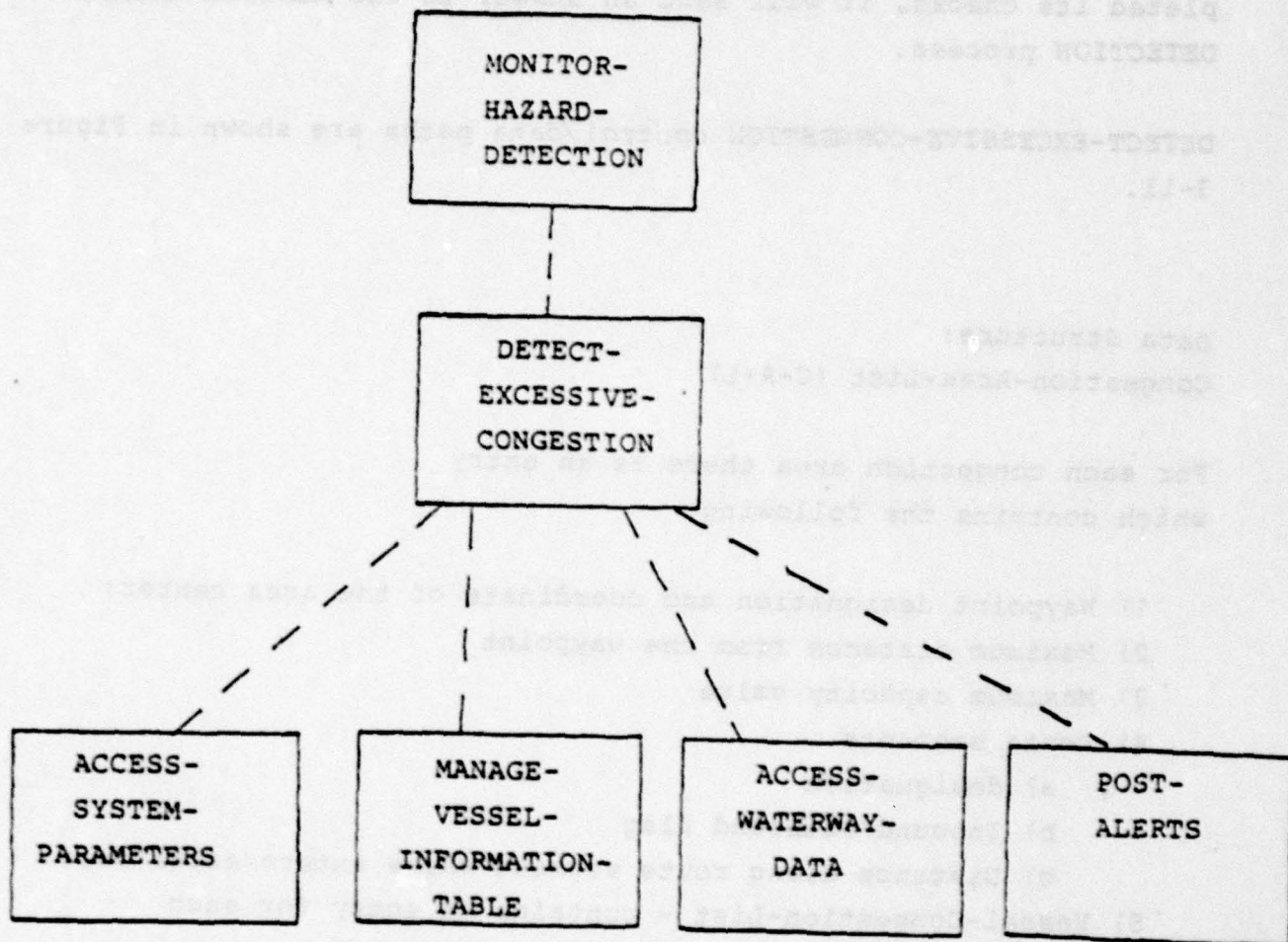


Figure 3-11. DETECT-EXCESSIVE-CONGESTION

Process DETECT-EXCESSIVE-CONGESTION

begin

while true do

begin

if no message is queued then wait for a message from  
the MONITOR-HAZARD-DETECTION process;

Send a message to the ACCESS-SYSTEM-PARAMETERS process  
requesting the excessive congestion look ahead time,  
the capacity contribution for each vessel size, and  
the cargo hazard parameters;

Wait for the answer;

for each entry in the Congestion-Area-List (C-A-L) do

begin

for vessel internal ID:= 1 to 3999 do

begin

Send a message to the MANAGE-VESSEL-INFORMATION-TABLE  
process requesting the VIT record with the  
internal ID;

Wait for the answer;

if the VIT record exists

and the Intended-Route-List in the VIT record is  
not empty

then

for each match between a segment in the Intended-  
Route-List and the route segment list in the  
C-A-L do

begin

Create an entry in the Vessel-Congestion-List  
(i.e., the vessel internal ID);

Get the vessel location and speed from the  
Position-Table;

Based on the vessel speed, location, intended  
route, the C-A-L route segment entry/exit  
points, and the look ahead time, compute the  
vessel entry and exit times;

Compute the vessel capacity contribution based  
on size and cargo hazard parameters;



```

        Place the vessel entry and exit times
        and capacity contribution in the Vessel-
        Capacity-Table (V-C-T) entry;
    end;
end;
for time=current time to look-ahead-time
step congestion-computation-interval do
begin
    Total:=0;
    for each entry in the Vessel-Congestion-
    Table do
        if entry-time  $\leq$  time  $\leq$  exit-time
        then
            Total:=Total+ Capacity contribution
            in the V-C-T entry;
        if Total  $\geq$  Capacity for Congestion Area
        then
            begin
                Send an Add-Alert message to the POST-
                ALERTS process;
                Wait for the answer;
                Get the alert-ID from the answer;
                Search the Congestion-Alert-List
                for the Alert-ID;
                if not already on the C-A-L
                then add the alert-ID and congestion-
                area-ID to the Congestion-Alert-List;
            end;
        else /*Total < Capacity*/
            begin
                Search the Congestion-Alert-List for the index to
                the C-A-L entry;
                if found
                then

```



```
begin
    Send a Delete-Alert message to the
    POST-ALERTS process;
    Remove the Congestion-Alert-List entry;
    Wait for the answer;

end;

end;

end;

end;

end;

end
```

### 3.2.7 PREDICT-DANGEROUS-ENCOUNTERS Process

The PREDICT-DANGEROUS-ENCOUNTERS process (see Figure 3-12) is used to detect when vessels are approaching a passage or overtaking in a channel too narrow to safely accomplish it, or when a vessel will be crossing the channel too close to an oncoming vessel in the channel. The location of each end point of up to 20 constricted areas will be specified in the data base. A preset time interval before a vessel will enter any constricted area, the subsystem will verify that the constricted area will be free of opposing traffic of sufficient size to be hazardous. Also, whenever a vessel is about to cross the channel, the movement will be accomplished at least a preset amount of time before or after any vessel in the channel arrives at the crossing location.

Each of the preset times mentioned above will be data base constants accessible to the Watch Supervisor. Also in the data base for each constricted area are specifications of the size combinations of vessels (one of four sizes for each vessel) which constitute a dangerous encounter. The PREDICT-DANGEROUS-ENCOUNTERS process will cycle at a rate not faster than once every 30 seconds. The rate, determined by the Watch Supervisor defined data base value, will be controlled by the MONITOR-HAZARD-DETECTION process.

PREDICT-DANGEROUS-ENCOUNTERS control/data paths are shown in Figure 3-13.



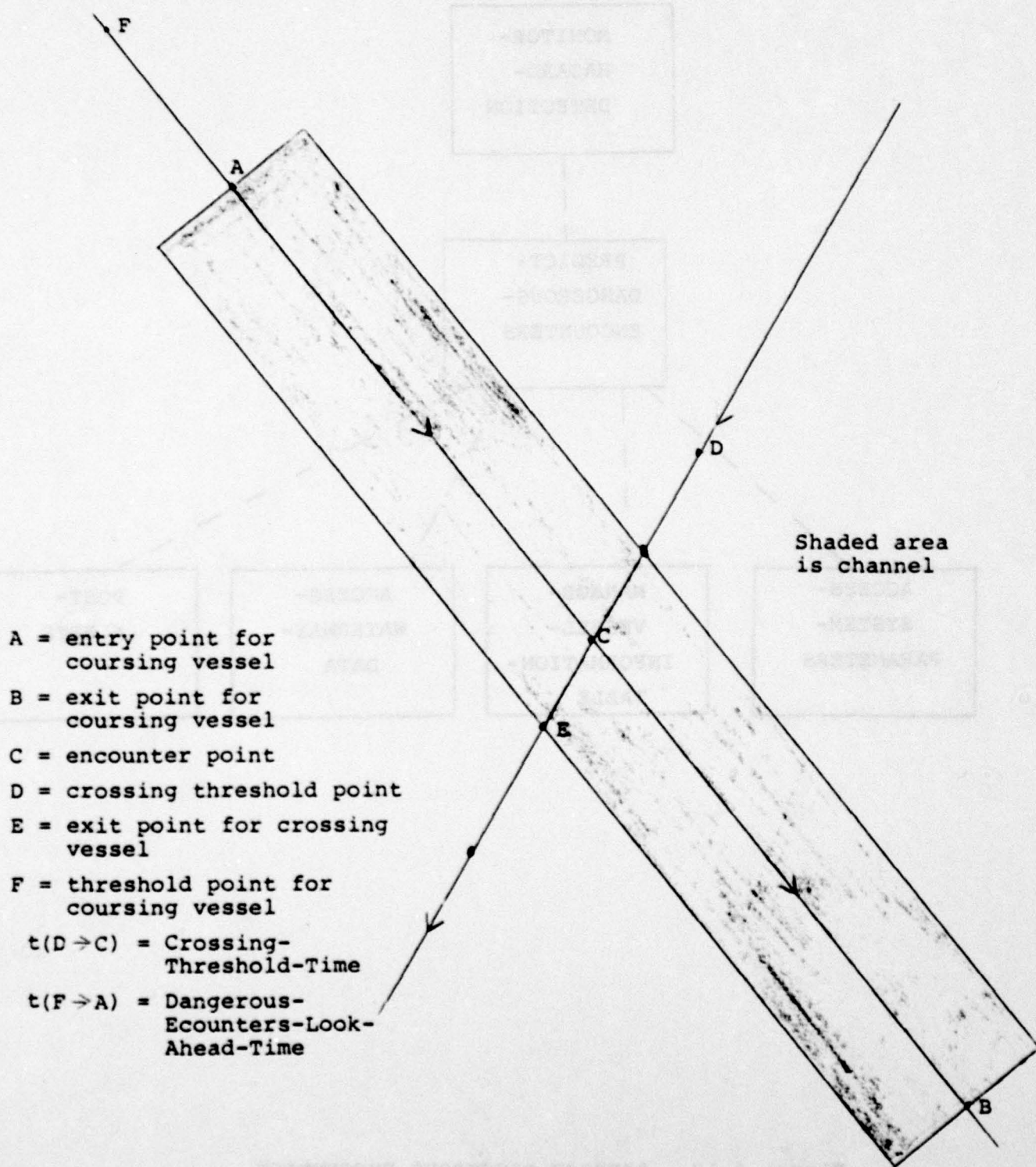


Figure 3-12. PREDICT-DANGEROUS-ENCOUNTERS



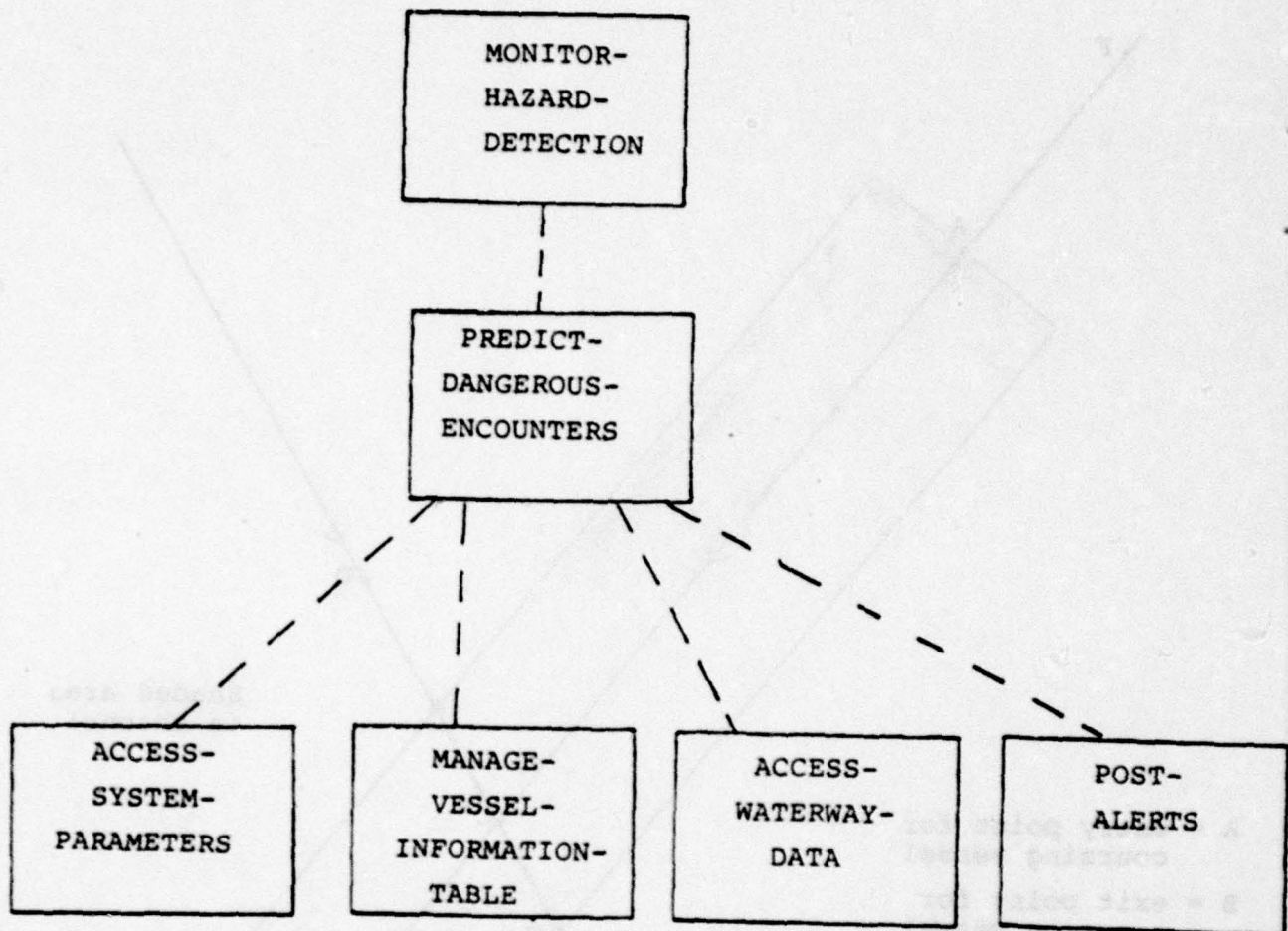


Figure 3-13. PREDICT-DANGEROUS-ENCOUNTERS

## Data Structures:

### Constricted-Channel-Table

This table consists of a maximum of 20 entries, 1 for each constricted channel in the waterway. Each entry consists of the following fields:

- 1) Designation of the route segment along which the constricted area lies.
- 2) The minimum channel width.
- 3) A flag indicating whether the channel allows traffic in one or both directions.
- 4) A list of the coordinates of the endpoints of the subsegments which the constricted area spans.
- 5) For every route segment which crosses the channel.
  - a) The coordinates of the 2 endpoints of the subsegment which crosses the channel.
  - b) The coordinates of the crossing point.
- 6) Channel-Vessel-List contains an entry for each identified, underway vessel with an intended route which, if followed, will cause it to enter or cross the constricted channel. This list is linked together in two ways: chronologically by entry/crossing time, and chronologically by exit time. Each entry contains the following fields:



- a) Link to next vessel entry in the entry/crossing time chain.
- b) Link to next vessel entry in the exit time chain.
- c) Vessel internal ID.
- d) Channel entry/crossing time.
- e) Channel exit time.
- f) Check-Time, i.e., time of day when vessel is approaching within threshold time of entering/crossing channel.
- g) Entering/Crossing flag.
- h) Direction flag
  - = +1 if vessel traveling same direction as order of route segment endpoints.
  - = -1 if vessel traveling opposite direction.
- i) Coordinates of entry/crossing point.

#### **Dangerous-Encounters-Alert-List**

This list contains an entry for each alert issued by this process to the POST-ALERTS process. Each entry contains the following fields:

- 1) Vessel internal ID.
- 2) Number of entry in Constricted-Channel-Table.
- 3) Alert number assigned by the POST-ALERTS process.



Process PREDICT-DANGEROUS-ENCOUNTERS

begin

while true do /\*infinite loop\*/

begin

Wait for a message from the MONITOR-HAZARD-DETECTION process;

Send a message to the ACCESS-SYSTEM-PARAMETERS process  
requesting the Dangerous-Encounters-Look-Ahead-Time, the  
Channel-Crossing-Threshold-Time Vessel-Size-Risk-Values  
and the Dangerous-Encounters-Size-Combination-Threshold;

Wait for the answer;

for each entry in the Constricted-Channel-Table do

begin /\*for each identified vessel\*/

for internal vessel ID: = 1 to 3999 do

begin

Send a message to the MANAGE-VESSEL-INFORMATION-  
TABLE process requesting the record with the specified  
vessel internal ID;

Wait for the answer;

if there is a VESSEL-INFORMATION-TABLE record with  
the specified internal vessel ID

and

the vessel is underway

and

the vessel is not exempt from dangerous-  
encounters processing

and

the vessel is following a route structure

and

the vessel's current and future portion of the  
intended route includes a route segment which is  
the route segment containing the constricted channel  
or a route segment which crosses the constricted  
channel

then

begin

Search Channel-Vessel-List for vessel internal ID;

```

        if not found
        then /*add vessel to Channel-Vessel-List*/
            ADD-TO-CHANNEL-VESSEL-LIST;
        else UPDATE-CHANNEL-VESSEL-LIST-ENTRY;
        end;
        else DELETE-VESSEL-FROM-LISTS
            end; /*for internal vessel ID: = 1 to 2999 do */
        end; /*for each entry in the Constricted-Channel-Table*/
        /*Now check vessels in the newly updated Constricted-
        Channel-Table for dangerous encounters*/
        for each entry in the Constricted-Channel-Table do
            begin /*check pairs of vessels for dangerous encounters*/
                N: = number of entries in Channel-Vessel-List;
                for i = 1 to N do
                    begin
                        Get vessel entry i from Channel-Vessel-List;
                        if Check-Time  $\leq$  current time of day
                        then
                            begin
                                for j = i+1 to N do
                                    begin
                                        Get vessel entry j from the Channel-Vessel-List;
                                        if Check-Time  $\leq$  current time of day
                                        and ENCOUNTER-TIME  $\neq$  0
                                        then ISSUE-ENCOUNTER-ALERT
                                        else CANCEL-ENCOUNTER-ALERT
                                    end;
                                end;
                            end;
                        end;
                    end;
                end; /*for each entry in the Constricted-Channel-Table*/
                Send an answer to the MONITOR-HAZARD-DETECTION process to
                notify it that the iteration is complete;
            end; /*while true*/
        end; /*process*/

```



Procedure ADD-TO-CHANNEL-VESSEL-LIST

begin

Determine available address at which to place Channel-Vessel-List entry and place following results in entry;

if vessel will be traveling through (coursing) channel /\*i.e., not crossing\*/

then

begin

Set coursing/crossing flag to coursing;

Set direction flag;

Determine coordinates of channel entry point;

Compute Channel-Entry-Time;

Compute Channel-Exit-Time;

Check-Time: = (Channel-Entry-Time) - (Dangerous-Encounters-Look-Ahead-Time);

end;

else /\*vessel will be crossing channel\*/

begin

Set coursing/crossing flag to crossing;

Determine coordinates of channel crossing point;

Compute Channel-Crossing-Time;

Check-Time: = (Channel-Crossing-Time) - (Crossing-Threshold-Time);

Channel-Exit-Time: = (Channel-Crossing-Time) + (Crossing-Threshold-Time);

end;

Link record chronologically into entry/crossing time chain;

Link record chronologically into exit time chain;

end;



Procedure UPDATE-CHANNEL-VESSEL-LIST-ENTRY

begin

if coursing/crossing flag = crossing /\*crossing channel\*/

then

begin

Based on present location, course, and speed (from  
Vessel-Information Table record), recompute Channel-  
Crossing-Time;

Check-Time: = (channel-Crossing-Time) - (Crossing-Threshold-Time);

Channel-Exit-Time: = (Channel-Crossing-Time) + (Crossing-  
Threshold-Time);

end;

else /\*vessel coursing through channel\*/

begin

Based on present location, course, and speed, recompute  
Channel-Entry-Time and Channel-Exit-Time;

Check-Time; + (Channel-Entry-Time) - (Dangerous-Encounters-  
Look-Ahead-Time);

end;

if Channel-Vessel-List entry is no longer in proper chronological  
position in entry/crossing time chain

then

begin

Remove from chain;

Relink in proper chronological chain position;

end;

if Channel-Vessel-List entry is no longer in proper chronological  
position in exit time chain

then

begin

Remove from chain;

Relink in proper chronological chain position;

end;

end;

Procedure DELETE-VESSEL-FROM-LISTS

begin

Search Channel-Vessel-List for vessel internal ID;

if found

then

begin

Unlink Channel-Vessel-List entry from entry/crossing time chain;

Unlink Channel-Vessel-List entry from exit time chain;

Zero fill entry;

end;

CANCEL-ENCOUNTER-ALERT

end;



Function ENCOUNTER-TIME

begin

if both vessels are coursing channel /\*neither crossing\*/

then

begin

Determine Vessel-Size-Risk based on the two vessel sizes;

if Vessel-Size-Risk  $\geq$  Dangerous-Encounters-Size-Combination-Threshold

then

if vessels proceeding in same direction

then

if an overtaking encounter will occur (based on dead reckoning along route)

then ENCOUNTER-TIME: = computed time of overtaking

else ENCOUNTER-TIME: = 0

else

if a passing (opposite directions) will occur

then ENCOUNTER-TIME: = Computed time of passing

else ENCOUNTER-TIME = 0

end;

end;



Procedure ISSUE-ENCOUNTER-ALERT

begin

Search of Dangerous-Encounters-Alert-List

for an entry containing the vessel internal ID and the  
current entry number of the Constricted-Channel-Table;

if found

then

begin

Send an Update-Alert message to the POST-ALERTS process  
containing the Alert Number and the update alert information;  
Wait for the answer;

end;

else

begin

Send an Add-Alert message to the POST-ALERTS process  
containing the new alert information;

Wait for the answer containing the Alert-Number assigned  
by the POST-ALERTS process;

Add an entry to the Dangerous-Encounters-Alert-List;

end;

end;

Procedure CANCEL-ENCOUNTER-ALERT

begin

Search the Dangerous-Encounters-Alert-List  
for an entry containing the vessel internal IDs and the  
current number of the Constricted-Channel Table;

if found

then

begin

Send a Delete-Alert message to the POST-ALERTS process  
with the Alert-Number contained in the Dangerous-Encounters-  
Alert-List entry;

Wait for an answer;

Delete the entry from the Dangerous-Encounters-Alert-List;

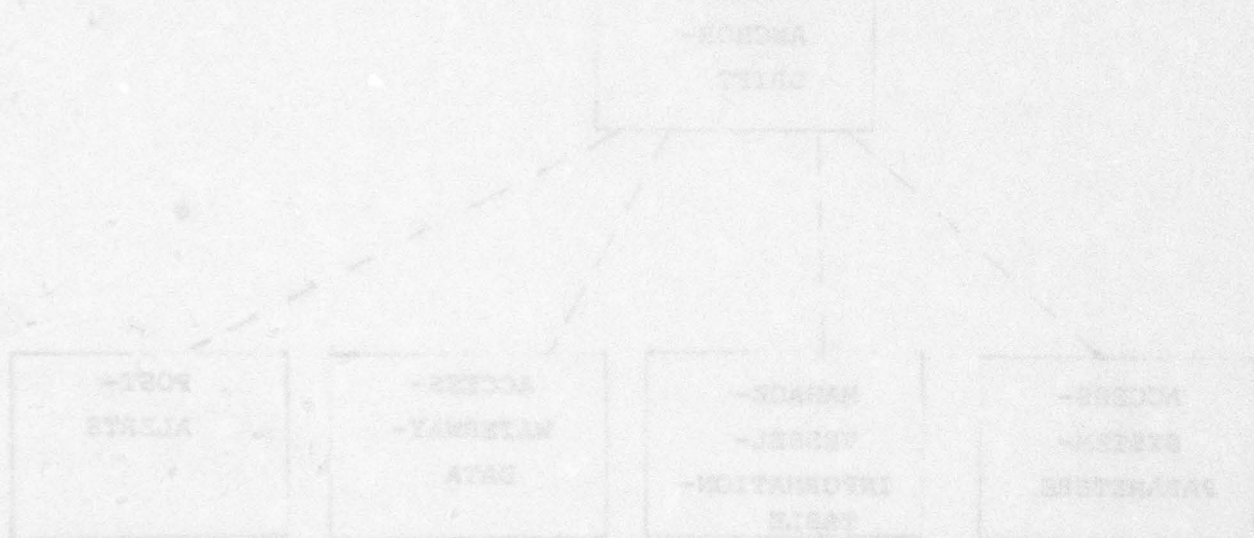
end;

end;

### 3.2.8 DETECT-ANCHOR-DRIFT Process

The DETECT-ANCHOR-DRIFT process will check the present measured position of all identified anchored vessels within the coverage of Level 4 or 5 sensors. An Anchor-Drift-Alert message will be sent to the POST-ALERTS process if a vessel moves outside the specified swing radius from its assigned anchorage location. This process cycles at a rate not faster than once per minute and is controlled by messages from the MONITOR-HAZARD-DETECTION process.

DETECT-ANCHOR-DRIFT control/data paths are shown in Figure 3-14.





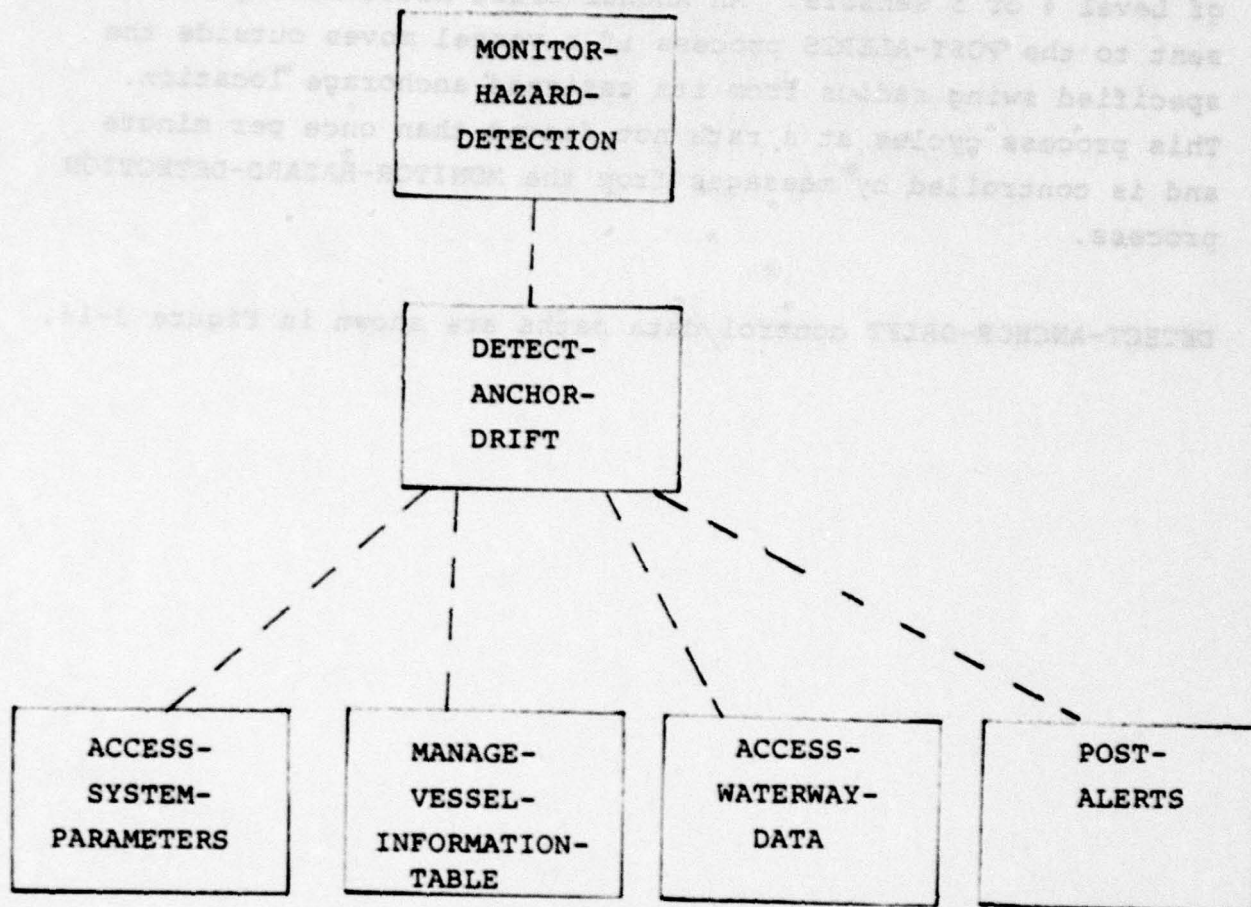


Figure 3-14. DETECT-ANCHOR-DRIFT

Process DETECT-ANCHOR-DRIFT

begin

while true do /\*infinite loop\*/

begin

Wait for a message from the MONITOR-HAZARD-

DETECTION process;

for vessel internal ID = 1 to 3999 do /\*for each identified  
vessel\*/

begin

Send a message to the MANAGE-VESSEL-INFORMATION-TABLE  
process requesting the V-I-T entry with the specified  
vessel internal ID;

Wait for the answer;

if there is currently an entry with the vessel internal ID  
then

begin

if vessel status = anchored

then

begin

Send a GET-P-R message to the ACCESS-PASSAGE-FILE  
process specifying the vessel internal ID;

Wait for the answer;

Get the swing radius and the anchorage location  
from the passage record;

if the vessel position (from the V-I-T record)  
is outside the swing radius

then

if vessel internal ID is not on the ANCHOR-  
DRIFT-ALERT-LIST

then

begin

Send an Add-Alert message to the POST-ALERTS  
process;

Wait for the answer;

Add the vessel internal ID and alert number  
to the A-D-A-L.

end;



else /\*vessel is inside swing radius\*/  
if vessel internal ID is on the A-D-A-L  
then

begin

Send a Delete-Alert message to the POST-ALERTS  
process with the alert number;

Wait for the answer;

Delete the entry for the A-D-A-L

end;

end;

Send an answer to the MONITOR-HAZARD-DETECTION process  
indicating that the if action is complete;

end;

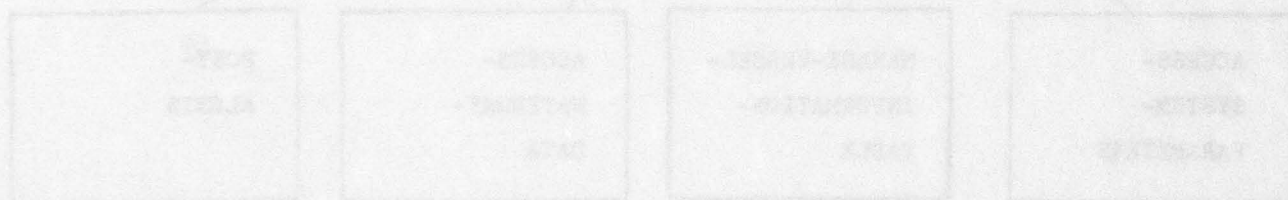
end;



### 3.2.9 DETECT-NAVAID-ADRIFT-MISSING Process

Each navaid designated by the Watch Supervisor which is within the coverage area of Level 4 or 5 sensors will be compared to its normal position. The normal display symbol for a navaid is replaced with a special symbol when Level 4 or 5 sensors lose track of a specified navaid (navaid missing), or if a navaid in track is outside its specified watch circle (navaid adrift). When a missing or adrift condition is detected, a message will be sent to the POST-ALERTS process which will in turn transmit a message to the MANAGE-ALERT-DISPLAY process in the appropriate display station processors. It will inform the MANAGE-MAP-DISPLAY process of the need for a special symbol. Processing to detect a navaid adrift/missing will be repeated at a rate not faster than once every minute as determined by the message from the MONITOR-HAZARD-DETECTION process.

DETECT-NAVAID-ADRIFT-MISSING control/data paths are shown in Figure 3-15.



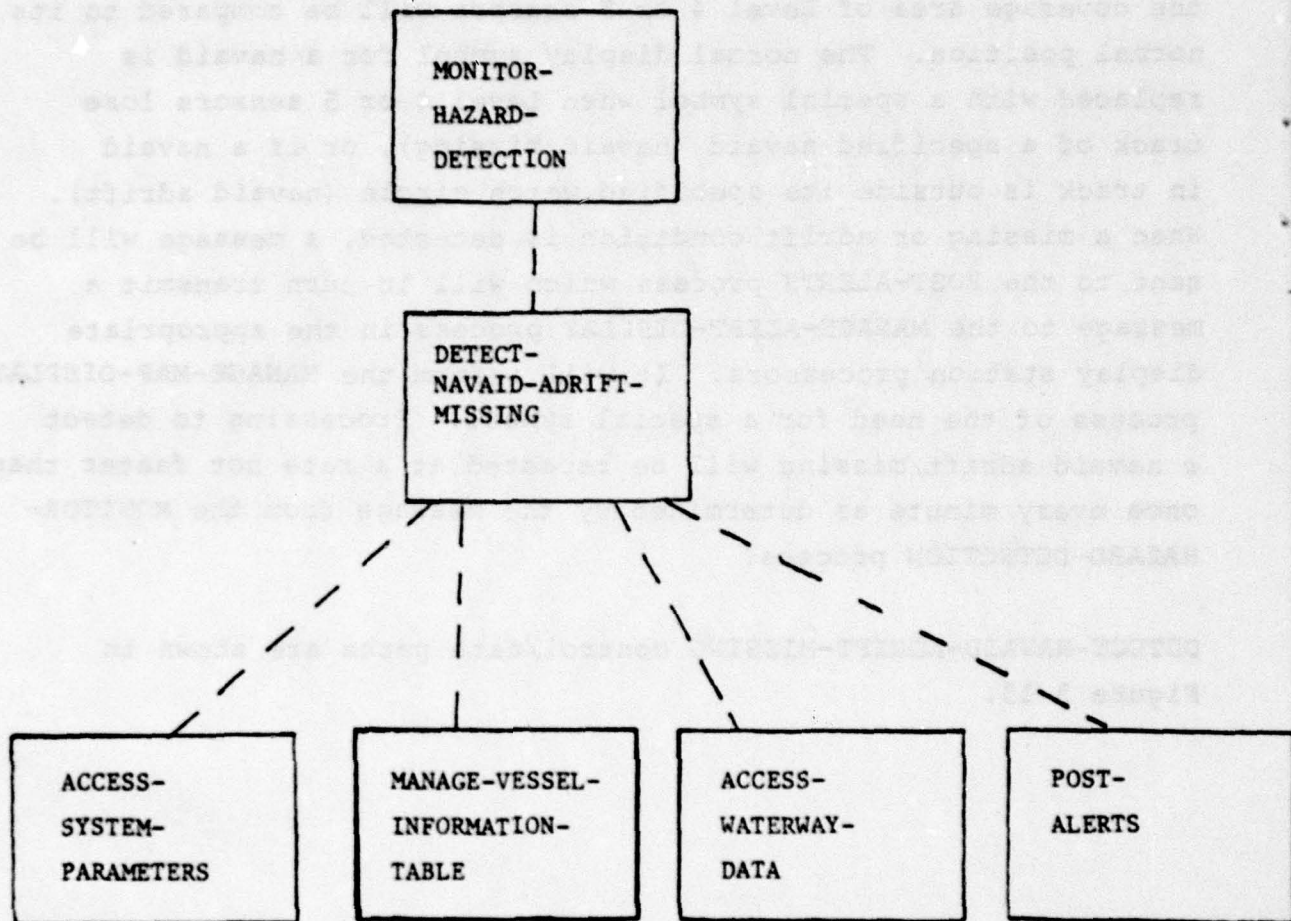


Figure 3-15. NAVAID-ADRIFT-MISSING



Process DETECT-NAVAID-ADRIFT-MISSING

begin

if no message is queued then

Wait for a message from the MONITOR-HAZARD-DETECTION  
process;

for internal ID=8000 to 10000 do /\*ID's of Navaid's\*

Read the entry in the Vessel-Information-Table (VIT)  
with the internal ID;

if the "tracked" flag is set and the flag indicating the  
navaid is exempt from adrift/missing processing is not  
set

then

begin

Send a Get-Navaid-Record message to the ACCESS-  
WATERWAY-DATA process specifying the navaid  
designation;

Wait for the answer;

Get the center and radius of the watch circle from  
the navaid record;

if the navaid location=0 /\*i.e., missing\*/

and the navaid designation is not on the  
Navaid-Missing-List (N-M-L)

then

begin

Send an Add-Alert message to the POST-ALERTS  
process;

Wait for the answer;

Add to the navaid to the N-M-L;

end;

else /\*not missing\*/

begin

if the navaid designation is on the N-M-L



```

then
  begin
    Send a Delete-Alert message to the POST-ALERTS
    process;
    Wait for the answer;
    Remove the navaid from the N-M-L;
  end;
if the navaid location is outside the watch circle radius
then
  if the navaid designation is on the Navaid-Adrift-List
  (N-A-L)
  then
    begin
      Send an Update-Alert message to the POST-ALERTS
      process;
      Wait for the answer;
    end;
  else /*not already on the N-A-L*/
    begin
      Send an Add-Alert message to the POST-ALERTS
      process;
      Wait for the answer;
      Add the navaid to the N-A-L with the alert ID
      assigned by the POST-ALERTS process.
    end;
  else; /*within watch circle*/
    if the navaid is on the N-A-L

```

then

begin

Send a Delete-Alert message to the POST-

ALERTS process;

Wait for the answer;

Delete the navaid from the N-A-L;

end;

end;

end;

Send the answer to the MONITOR-HAZARD-DETECTION process;

end;

end;



#### 3.2.10 DETECT-EXCESSIVE-VESSEL-SPEED Process

The DETECT-EXCESSIVE-VESSEL-SPEED process will check the measured speed of vessels in all route segments and "cells" for which a maximum speed has been specified. If a vessel is proceeding at a speed "x" knots above the limit (where "x" is a data base constant accessible by the Watch Supervisor), a message will be sent to the POST-ALERTS process. This process will cycle at a rate not faster than once every minute and will be controlled by messages from the MONITOR-HAZARD-DETECTION process.

DETECT-EXCESSIVE-VESSEL-SPEED control/data paths are shown in Figure 3-16.

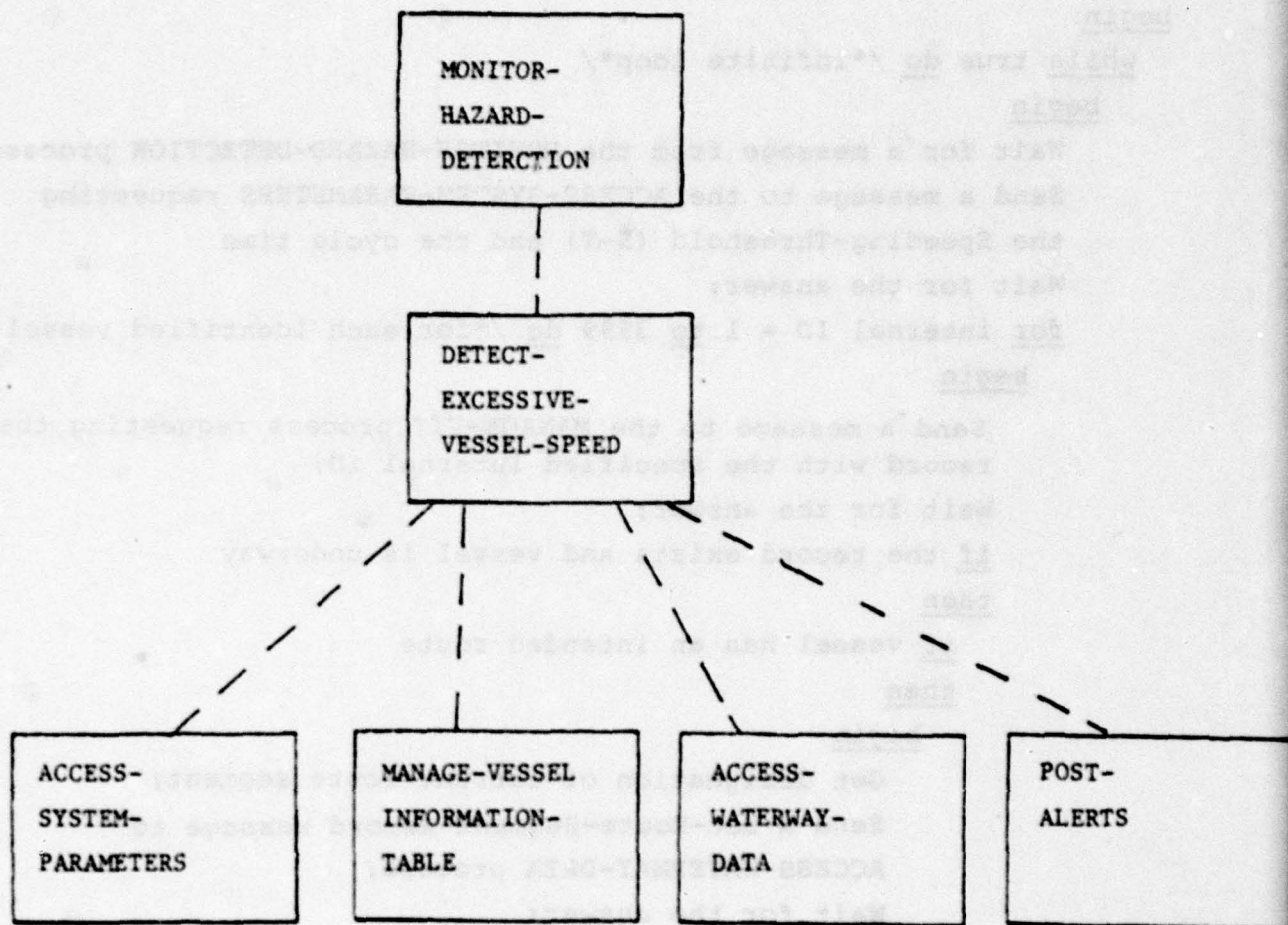


Figure 3-16. DETECT-EXCESSIVE-VESSEL-SPEED



Process DETECT-EXCESSIVE-VESSEL-SPEED

begin

while true do /\*infinite loop\*/

begin

Wait for a message from the MONITOR-HAZARD-DETECTION process;

Send a message to the ACCESS-SYSTEM-PARAMETERS requesting  
the Speeding-Threshold (S-T) and the cycle time

Wait for the answer;

for internal ID = 1 to 3999 do /\*for each identified vessel\*/

begin

Send a message to the MANAGE-VIT process requesting the  
record with the specified internal ID;

Wait for the answer;

if the record exists and vessel is underway

then

if vessel has an intended route

then

begin

Get designation of current route segment;

Send a Get-Route-Segment-Record message to  
ACCESS-WATERWAY-DATA process;

Wait for the answer;

SL: = Get speed limit from route segment record;

if vessel speed > SL + ST

then

begin

if vessel on Speeding-Alert-List (S-A-L)

then

begin

Send Update-Alert message to the POST-

ALERTS (P-A) process with the current speed;

Wait for answer

end

else

```

    begin
        Send Add-Alert message to the P-A process;
        Wait for answer;
        Add to S-A-L;
    end
end
else /*not speeding*/
    if on S-A-L
    then
        begin
            Send a Delete-Alert message to P-A;
            Wait for answer;
            Remove from S-A-L
        end
    end
else /*not following a route*/
    determine current cell based on vessel
    location
    begin
        Send a Get-Cell-Data message to ACCESS-WATERWAY-
        DATA process;
        Wait for the answer;
        if cell-speed-limit = 0
        or cell-speed-limit + ST < vessel speed
        then
            if on S-A-L
            then
                begin
                    Send Delete-Alert message to P-A;
                    Wait for answer;
                    Remove from S-A-L
                end
            end
        else /*speeding in cell*/
            if on S-A-L
            then
                Send Update-Alert message to P-A
                Wait for answer
            end
        end
    end

```

```
else
  begin
    Send an Add-Alert message to P-A;
    Wait for answer;
    Add to S-A-L;
  end
end /*for internal ID*/
Send answer to MONITOR-HAZARD-DETECTION;

end;
end;
```



### 3.3 ALERT POSTING

This section describes the POST-ALERTS process which resides in the main processor. It receives messages (i.e., Enter-Alert, Update-Alert, or Delete-Alert) from the hazard detection processes instructing it to enter, modify, or delete an entry on the Master-Alert-Queue. The process determines which display station to notify, based on the location of the hazard, and which display station is assigned to the sector covering the location. The process then sends a message to the MANAGE-ALERT-DISPLAY process (see section 4.3) at the appropriate watchstander display station, and at the Watch Supervisor display station. This message contains information concerning the alert, and whether the alert is being added, deleted, or modified.

Relationship to Functional Description:

Appendix 8: 2.3.2, 5.6.2.a.

Data Structure:

The Master-Alert-Queue is a singly linked queue consisting of one entry for each alert generated by a hazard detection process. Each alert entry contains the following data: 1) Alert Number (unique identifier assigned by the POST-ALERTS process), 2) The type of alert, 3) The time remaining for effective action, 4) The waterway sector containing the alert, 5) Alert status, and 6) All other information describing the alert hazard.

### 3.4 LOGGING

The logging process will provide support for the logging functions carried out by the subsystem. A family of processes will be used, with one member supporting each type of log.

Specifically, MAINTAIN-OPERATIONS-LOG, SAVE-MANUAL-BACKUP-DATA, MAINTAIN-TRAFFIC-SUMMARIES and MAINTAIN-PASSAGE-HISTORY processes will manage the logging functions.

The appropriate logging process will receive messages from processes needing to enter data into that particular log. The logging process will then enter the data into a buffer and call the operating system to write them to the logging medium when the buffers are full.

#### 3.4.1 MAINTAIN-OPERATIONS-LOG Process

The MAINTAIN-OPERATIONS-LOG process will manage the entry of data into the Operations Log.

MAINTAIN-OPERATIONS-LOG control/data paths are shown in Figure 3-17.

#### Message Types:

The process consists of a set of procedures, each of which performs the function requested by one of the following message types.



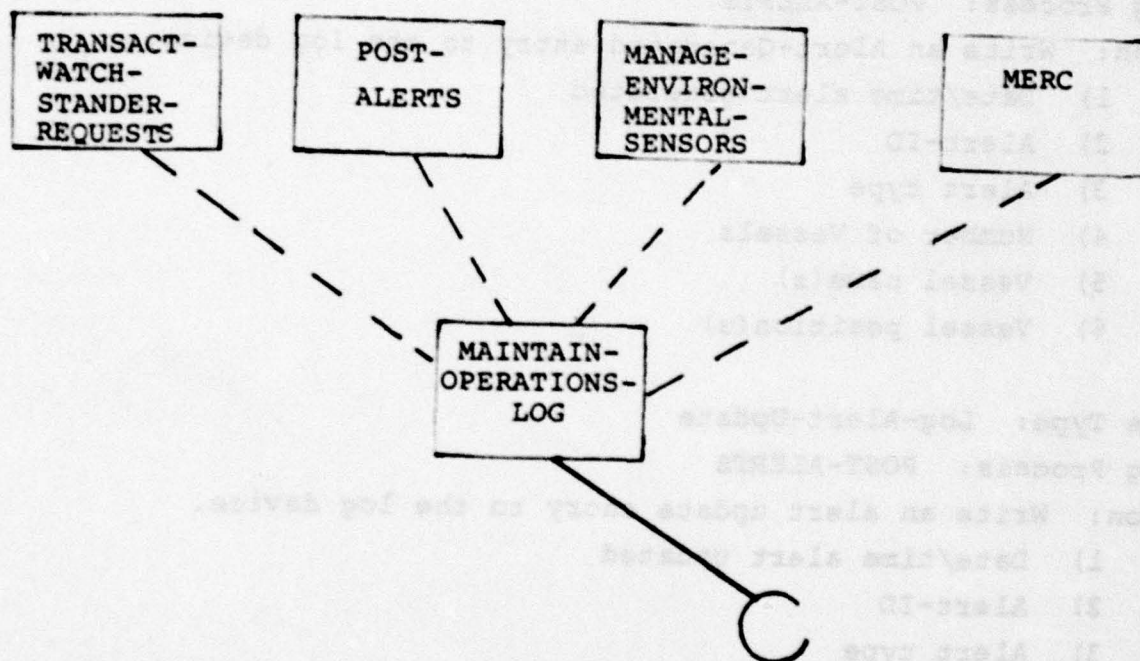


Figure 3-17. MAINTAIN-OPERATIONS-LOG



Message Type: Log-Non-Standard-Route

Sending Process: TRANSACT-WATCHSTANDER-REQUESTS

Function: Write non-standard (temporary) route segment record  
onto logging device;

Input: 1) Data/time route segment created  
2) Route Segment Designation  
3) Vessel Name  
4) Watchstander-ID  
5) Route Segment Attributes

Message Type: Log-Alert-Generation

Sending Process: POST-ALERTS

Function: Write an Alert-Generated entry to the log device.

Input: 1) Date/time alert generated  
2) Alert-ID  
3) Alert type  
4) Number of Vessels  
5) Vessel name(s)  
6) Vessel position(s)

Message Type: Log-Alert-Update

Sending Process: POST-ALERTS

Function: Write an alert update entry to the log device.

Input: 1) Date/time alert updated  
2) Alert-ID  
3) Alert type  
4) Number of Vessels  
5) Vessel name(s)  
6) Vessel position(s)

Message Type: Log-Alert-Response

Sending Process: TRANSACT-WATCHSTANDER-REQUESTS

Function: Write an Alert-Response entry to the log device

Input: 1) Date/time of alert response  
2) Alert-ID  
3) Alert type  
4) Response type (cancel, realert, hold, or restore)

Message Type: Log-Vessel-Position

Sending Process: MANAGE-VESSEL-INFORMATION-TABLE

Function: Write the vessel position to the log device

Input: 1) Date/time of position  
2) Vessel name  
3) Vessel position

Message Type: Log-System-Fault

Sending Process: MERC

Function: Write the system fault entry to the log device

Input: 1) Date/time of system fault  
2) Name of faulty function  
3) Type of fault  
4) Fault information

Message Type: Log-Supervisor-Decision

Sending Process: TRANSACT-WATCHSTANDER-REQUESTS

Function: Log decision of supervisor on questionable watch-  
stander input.

Input: 1) Date/time of decision  
2) Watch supervisor ID  
3) Watchstander ID  
4) Vessel name  
5) Questioned data  
6) Decision



Message Type: Log-Sensor-Data

Sending Process: MANAGE-ENVIRONMENTAL-SENSORS

Function: Log averaged weather and current/tide data to the log device.

- Input:
- 1) Date/time data averaged
  - 2) Number of Weather Sensor Stations
  - 3) For each station
    - a) Station designation
    - b) Number of readings used to compute average data
    - c) Temperature
    - d) Visibility
    - e) Precipitation rate
    - f) Wind speed
    - g) Wind direction
  - 4) Number of current/tide stations
  - 5) For each station
    - a) Station designation
    - b) Number of readings
    - c) Current speed
    - d) Current designation
    - e) Tide level
    - f) Average wave height



Process MAINTAIN-OPERATIONS-LOG

begin

while true do

begin

Wait for a message;

Send an answer acknowledging receipt of the message;

Get a buffer in which to place data to be output to the logging device;

case message type of

Log-Non-Standard-Route: COPY-ROUTE-DATA

Log-Alert-Generation: COPY-ALERT-GENERATION-DATA

Log-Alert-Update: COPY-ALERT-UPDATE-DATA

Log-Alert-Response: COPY-ALERT-RESPONSE-DATA

Log-Vessel-Position: COPY-VESSEL-POSITION-DATA

Log System Fault: COPY-SYSTEM-FAULT-DATA

Log-Supervisor-Decision: COPY-SUPERVISOR-DECISION-DATA

Log-Sensor-Data: COPY-SENSOR-DATA

end;

Request the operating system to write the contents of the output buffer to the logging device;

end;

end;

Procedure COPY-ROUTE-DATA

begin

Copy the date/time, the route segment designation, the watchstander ID, and the vessel name from the message to the output buffer;

end;

procedure COPY-ALERT-GENERATION-DATA

begin

Copy the date/time the alert-ID, the alert type, the number of vessels involved, the vessel name(s) and the vessel position(s) to the output buffer;

end;

procedure COPY-ALERT-UPDATE-DATA

begin

Copy the date/time the alert-ID, the alert type, the vessel name(s) and the vessel position(s) from the message to the output buffer;

end;

procedure COPY-ALERT-RESPONSE-DATA

begin

Copy the date/time, the name of the faulty function, the fault type, and other fault information from the message to the output buffer;

end;

procedure COPY-SUPERVISOR-DECISION-DATA

begin

Copy the date/time, the watchstander ID, the watch supervisor ID, the vessel name, the questioned data, and the watch supervisor's decision, from the message to the output buffer;

end;



procedure COPY-SENSOR-DATA

begin

Copy the date/time, the number of weather sensor stations,  
and for each station:

- a) weather state designation
- b) temperature
- c) visibility
- d) precipitation rate
- e) wind speed
- f) wind direction

Copy the number of current/tide sensor station, and for each  
station:

- a) station designation
- b) current speed
- c) current direction
- d) average tide level
- e) average wave height

end;

procedure COPY-SYSTEM-FAULT-DATA

begin

Copy the date/time of system fault, the name of the faulty  
function, the type of fault and the other fault information  
from the message to the output buffer;

end;

procedure COPY-VESSEL-POSITION-DATA

begin

Copy the date/time of the position, the vessel name and the  
vessel position from the message to the output buffer;

end;

### 3.4.2 SAVE-MANUAL-BACKUP-DATA Process

The SAVE-MANUAL-BACKUP-DATA process will maintain a log of the manual backup data.

SAVE-MANUAL-BACKUP-DATA control/data paths are shown in Figure 3-18.

Data structures for this process include:

- . The Landmark-Position-List contains an entry for each landmark. Each entry consists of 1) the landmark name, and 2) the coordinates of the landmark.
- . The Vessel-Sector-List-Array consists of a list of internal vessel ID's for each sector.



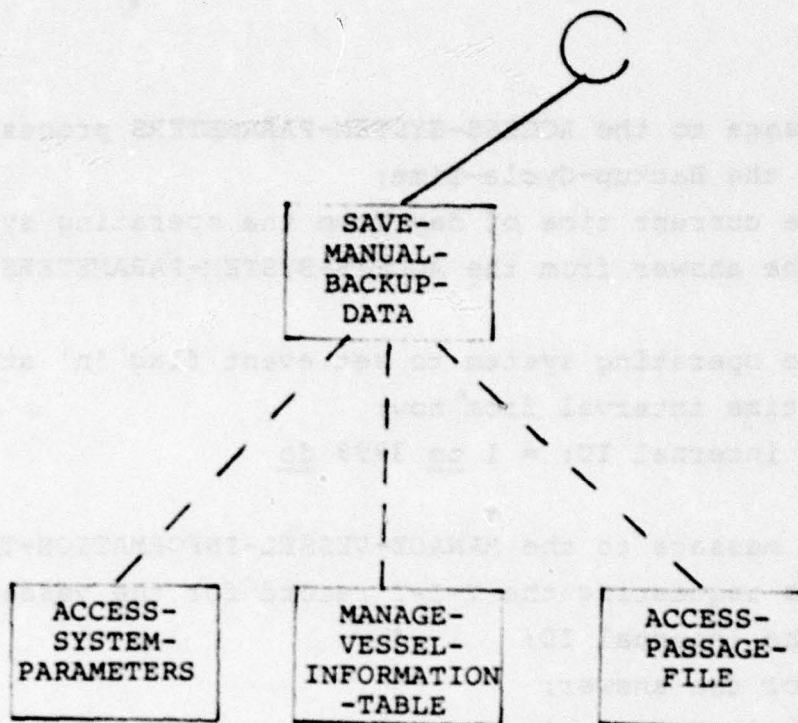


Figure 3-18. SAVE-MANUAL-BACKUP-DATA

Process SAVE-MANUAL-BACKUP-DATA

begin

while true do

begin

Send a message to the ACCESS-SYSTEM-PARAMETERS process  
requesting the Backup-Cycle-Time;

Request the current time of day from the operating system;

Wait for the answer from the ACCESS-SYSTEM-PARAMETERS  
process;

Request the operating system to set event flag 'n' at  
the cycle time interval from now;

for vessel internal ID: = 1 to 3999 do

begin

Send a message to the MANAGE-VESSEL-INFORMATION-TABLE  
process requesting the V-I-T record for the vessel  
with the internal ID;

Wait for the answer;

if such a record exists

then

Determine the sector from the vessel position in  
V-I-T;

Add the internal ID to the Vessel-Sector-List for  
the sector;

end;

for each Sector list in the Vessel-Sector-List-Array do  
SAVE-SECTOR-VESSEL-DATA (sector);

if event flag 'n' is set

then send a message to the LERC that this process is  
running behind schedule;

else Wait for event flag 'n' to be set;

end;

end;



Procedure SAVE-SECTOR-VESSEL-DATA (Sector)

begin

for each internal-id for the specified sector in the Vessel-Sector-List-Array-List do

begin

Send a message to the ACCESS-PASSAGE-FILE process requesting the passage record for the vessel with the specified internal-ID;

Wait for the answer;

Copy the vessel name, vessel type, vessel cargo, origin of passage, and destination of passage from the passage record to the output buffer for the backup device;

if the intended route list in the passage record is empty then

begin

Send a Get-Reporting-Point-Record to the ACCESS-PASSAGE-FILE process with the disc address of the latest reporting point record;

Wait for the answer;

Copy the last reporting point designation, the date/time of passage and the next reporting point designation, estimated date/time for that reporting point from the reporting point record to the backup device output buffer;

end;

else

begin

Send a message to the MANAGE-VESSEL-INFORMATION-TABLE requesting the V-I-T record with the internal ID;

Wait for the answer;

Search the Landmark-Position-List for the landmark closest to the vessel position;

Compute the range and bearing from the landmark to the vessel;

Move the range and bearing into the output buffer;

Copy the vessel course and speed from the V-I-T record to the output buffer;

Request the operating system to write the output buffer to the backup device;

end;

end;

end;



### 3.4.3 MAINTAIN-TRAFFIC-SUMMARIES process

The MAINTAIN-TRAFFIC-SUMMARIES process will manage the entry of data into the Traffic Summaries Log.

MAINTAIN-TRAFFIC-SUMMARIES control/data paths are shown in Figure 3-19.

Data structures for this process include:

- . Summary-Date is a string 'mm/dd/yy' as will be printed at the top of the traffic summary report.
- . Type-Count is an array containing an element for each vessel-type. Each element contains the number of passages of vessels of that type.
- . Dangerous-Cargo-Count contains the number of passages with dangerous cargoes.
- . Alert-Count (alert-type) is an array containing an element for each type of hazard alert. Each element contains the number of alerts of the corresponding type.

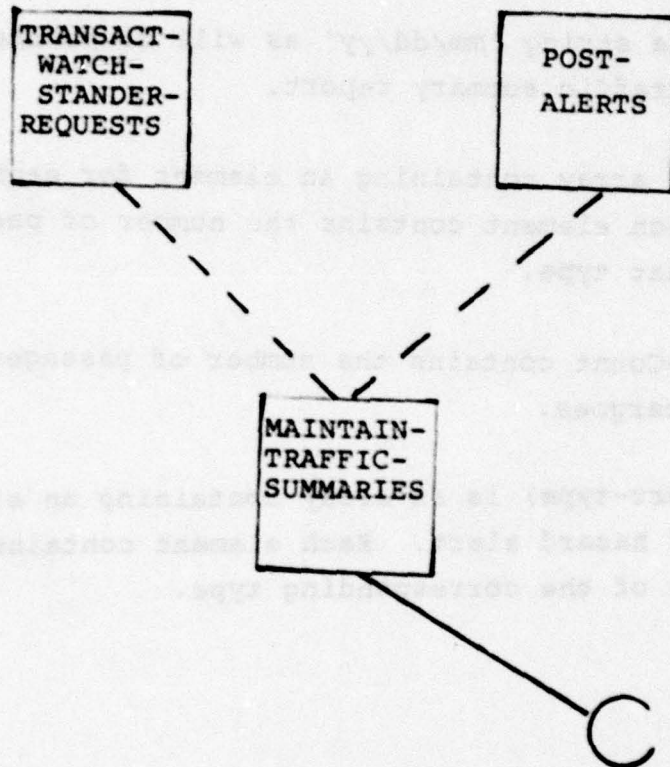


Figure 3-19. MAINTAIN-TRAFFIC-SUMMARIES



Process MAINTAIN-TRAFFIC-SUMMARIES

begin

SCHEDULE-NEXT-REPORT

while true do

begin

Wait for a message or report-event-flag to be set;

if report-event-flag is set

then /\*produce report\*/

begin

SCHEDULE-NEXT-REPORT;

Get a buffer for the traffic summary report;

Move the Summary-Date to the report output buffer;

Total-Passage-Count: = 0;

for each vessel-type do

begin

Total-Passage-Count: = (Total-Passage-Count)  
+ (Type-Count (vessel-type));

Format and move the Type-Count (vessel-type) to  
the output buffer;

end;

Format and move the Total-Passage-Count to the  
output buffer;

Format and move the Dangerous-Cargo-Count to the  
output buffer;

for each alert-type do

Format and move Alert-Count (alert-type) to the  
output buffer;

Request the operating system to write the output  
buffer to the line printer;

Summary-Date: = today's date;

Zero all counters;

end;

```

else /*process a message*/
  begin
    case message-type of
      Passage-Msg.:
        begin
          Increment Type-Count (vessel type);
          if Dangerous-Cargo-Flag is set in the message
            then Increment Dangerous-Cargo-Count;
          end;
          Alert-Msg.: Increment Alert-Count (alert type);
          end;
        end;
        Send an answer acknowledging message processed;
      end;
    end;
  end;

```



Procedure SCHEDULE-NEXT-REPORT

begin

Request the current time of day from the operating system;

Compute the time interval from now to midnight;

Request the operating system to set the report-event-flag  
at the computed time interval from now;

end;

#### 3.4.4 MAINTAIN-PASSAGE-HISTORY Process

The MAINTAIN-PASSAGE-HISTORY process will manage the logging of data into the Vessel Passage History File.



Process MAINTAIN-PASSAGE-HISTORY

begin

While true do

begin

Wait for a message;

Send an answer acknowledging receipt of the message;

/\*the message will contain a copy of the passage record,  
reporting point records, and temporary route segment  
records\*/

if the vessel status is changing from underway

then

begin

Copy the vessel name, vessel type, vessel cargo,  
origin point, origin date/time, destination point,  
destination date/time, and number of barges (if  
vessel type is tug or two boat) from the received  
message to the history file output buffer;

while not end of reporting point record chain do

begin

Get the next (most recent) reporting point record  
in the chain;

if a route segment was entered at this reporting  
point

then

begin

Copy the route segment designation and the  
reporting point date/time from the reporting  
point record in the message to the history  
file output buffer;

if the route segment is temporary

then Copy the temporary route segment description  
to the history file output buffer;

end;

end;

end;

if the vessel status is changing from anchored  
then

begin

Copy the vessel name, type, cargo, anchorage  
designation, anchorage location, anchorage  
date/time, and status change (from anchored)  
date/time from the message to the history file  
output buffer;

end;

Request the operating system to write the history  
file buffer to the history file;

end;

end;



### 3.5 ENVIRONMENTAL SENSORS

The MANAGE-ENVIRONMENTAL-SENSORS process will receive data from environmental sensors and enter the data into the system data base.

MANAGE-ENVIRONMENTAL-SENSORS control/data paths are shown in Figure 3-20.

#### Data Structures:

Automatic-Weather-Station-Data-Accumulation-Table consists of a count of the number of stations and an entry for each station. Each entry consists of the following fields:

- 1) Station Designation
- 2) Data Count--number of times data received from the station
- 3) Cumulative Temperature
- 4) Cumulative vsb
- 5) Cumulative prec. rate
- 6) Cumulative wind speed
- 7) Cumulative wind dir

Current-Tide-Station-Data-Accumulation-Table consists of a count of the number of stations and an entry for each station. Each entry consists of the following fields:

- 1) Station Designation
- 2) Data Count
- 3) Cumulative Current Speed
- 4) Cumulative Current Direction
- 5) Cumulative Current Tide Level
- 6) Cumulative Wage Height

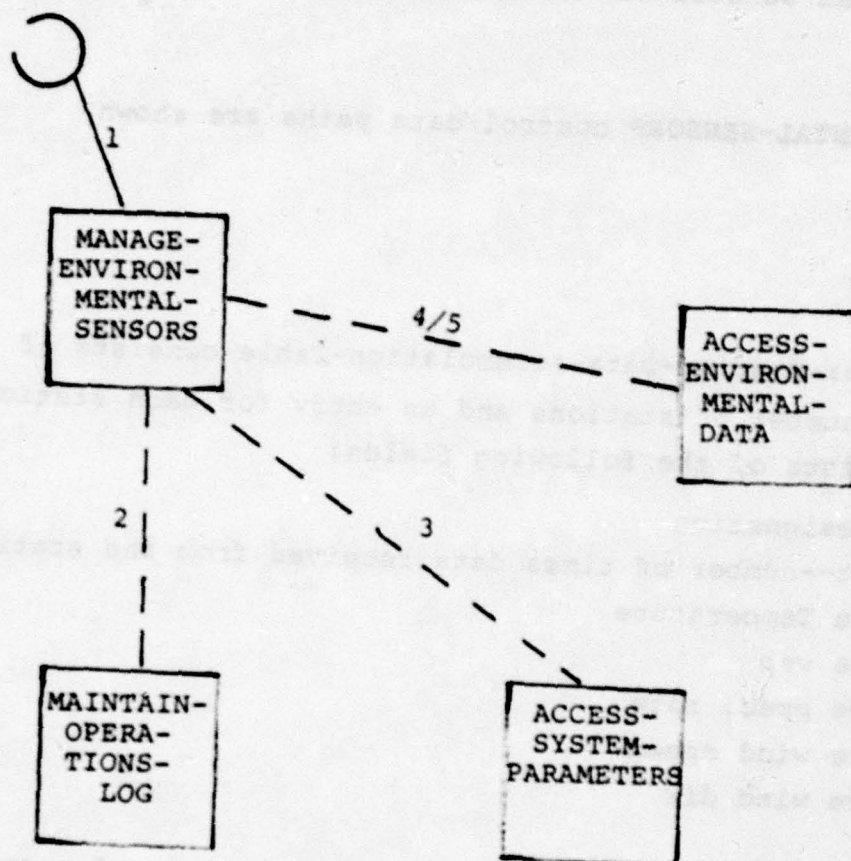


Figure 3-20. MANAGE-ENVIRONMENTAL-SENSORS



FIGURE 3-20. MANAGE-ENVIRONMENTAL-SENSORS

- 1) Data from input device handlers
- 2) Log-Sensor-Data message
- 3) Message requesting SENSOR-LOGGING-CYCLE-TIME
- 4) Message to Modify-Automatic-Weather-Record
- 5) Message to Modify-Current-Tide-Record

Process MANAGE-ENVIRONMENTAL-SENSORS

begin

Zero accumulation fields and data-counts;

while true do

begin

Send a message to the ACCESS-SYSTEM-PARAMETERS process  
requesting the Sensor-Logging-Cycle-Time (SLCT);

Request the operating system to set the delay-event  
flag at the SLCT interval from now;

Wait for the answer;

Case event-flag of

delay-event flag: LOG-SENSOR-DATA

weather-data-event-flag: ACCUMULATE-WEATHER-DATA

current-tide-event-flag: ACCUMULATE-CURRENT-TIDE-DATA

end;

end;

end;



Procedure LOG-SENSOR-DATA

begin

Clear the delay-event-flag;

Request the operating system to set the delay-event flag  
at the SLCT interval from now;

Request the present date/time from the operating system;

for each environmental sensor station do /\*for all weather and  
current/tide stations\*/

begin

Compute average readings by dividing each data accumulation  
field by the data count (number of sets of readings)  
for the station;

Move the station designation, the data count, and the  
average readings to the message buffer;

if station is a weather station

then Format and send a Modify-Automatic-Weather-Record  
message to the ACCESS-ENVIRONMENTAL-DATA process to  
place the average data on disc;

else Format and send a Modify-Current-Tide-Record message  
to the ACCESS-ENVIRONMENTAL-DATA process to the average  
data on disc;

Wait for the answer;

Zero the data count and the accumulation fields for the  
station;

end;

Move the present date/time to the message buffer;

Move the number of weather sensor stations to the message buffer;

Move the number of current/tide sensor stations to the message  
buffer;

Send the Log-Sensor-Data message (containing the above data)  
to the MAINTAIN-OPERATIONS-LOG process;

end;

procedure        ACCUMULATE-WEATHER-DATA

begin

    Receive the weather data input buffer from the  
    sensor input device handler;  
    Get the weather station designation from the input  
    buffer;  
    Increment the data-count for the station;  
    Add each data item value in the input buffer  
    to its corresponding accumulation field;  
    Clear the weather-data-event-flag;

end

procedure        ACCUMULATE-CURRENT-TIDE-DATA

begin

    Receive the current/tide data input buffer from  
    the sensor input device handler;  
    Get the current/tide station designation from the  
    input buffer;  
    Increment the data-count for the station;  
    Add each data item value in the input buffer to its  
    corresponding accumulation field;  
    Clear the current-tide-event-flag;

end



The processes which interface between the watchstander (or Watch Supervisor) and the other applications processes reside in the display station processor.

These processes are TRANSACT-WATCHSTANDER-REQUESTS, MANAGE-MAP-DISPLAY, MANAGE-ALERT-DISPLAY, and MANAGE-ACTION-REQUIRED-LIST.

The TRANSACT-WATCHSTANDER-REQUESTS process is responsible for coordinating 1) the input from the standard and function keyboards, 2) the output of prompts, forms, and results on the alphanumeric display screen, and 3) the communication with other processes (where necessary). The TRANSACT-WATCHSTANDER-REQUESTS process contains the logic necessary to perform this coordination for any of the functions which may be requested by a watchstander.

The MANAGE-MAP-DISPLAY maintains the map display and the display processor table of vessel and buoy positions.

The MANAGE-ALERT-DISPLAY is responsible for maintaining and displaying (on the alert display screen) the list of alerts queued to the display station.

The MANAGE-ACTION-REQUIRED-LIST process is responsible for maintaining the list of messages queued to the watchstander awaiting his acknowledgement, and possible required actions.

#### 4.1 TRANSACT-WATCHSTANDER-REQUESTS Process

Brief Description: This process obtains input from the watchstander keyboard via the GET-KEY procedure. Based on the function specified, the appropriate procedure (subroutine) is called. Within the called procedure, forms are output to the alphanumeric display, using DISPLAY-ALPHANUMERIC, and watchstander supplied data is input and verified. When required, a message is sent to the appropriate procedure to perform data base accesses and/or calculations. The process then waits for an answer. When an answer is received, it is formatted and output to the alphanumeric display. The process then waits for another watchstander input from the keyboard of the display station.

TRANSACT-WATCHSTANDER-REQUESTS control/data paths are shown in Figure 4-1.

Relationship to Functional Description:

Appendix 8 -

- 3.6.1 I/O Equipment
- 4.1 Watchstander Inputs
- 5.5.1 Manually Input Data
- 5.6.1 Demand Processing

Frequency of Use: Frequency at which the watchstander requests functions from the keyboard.

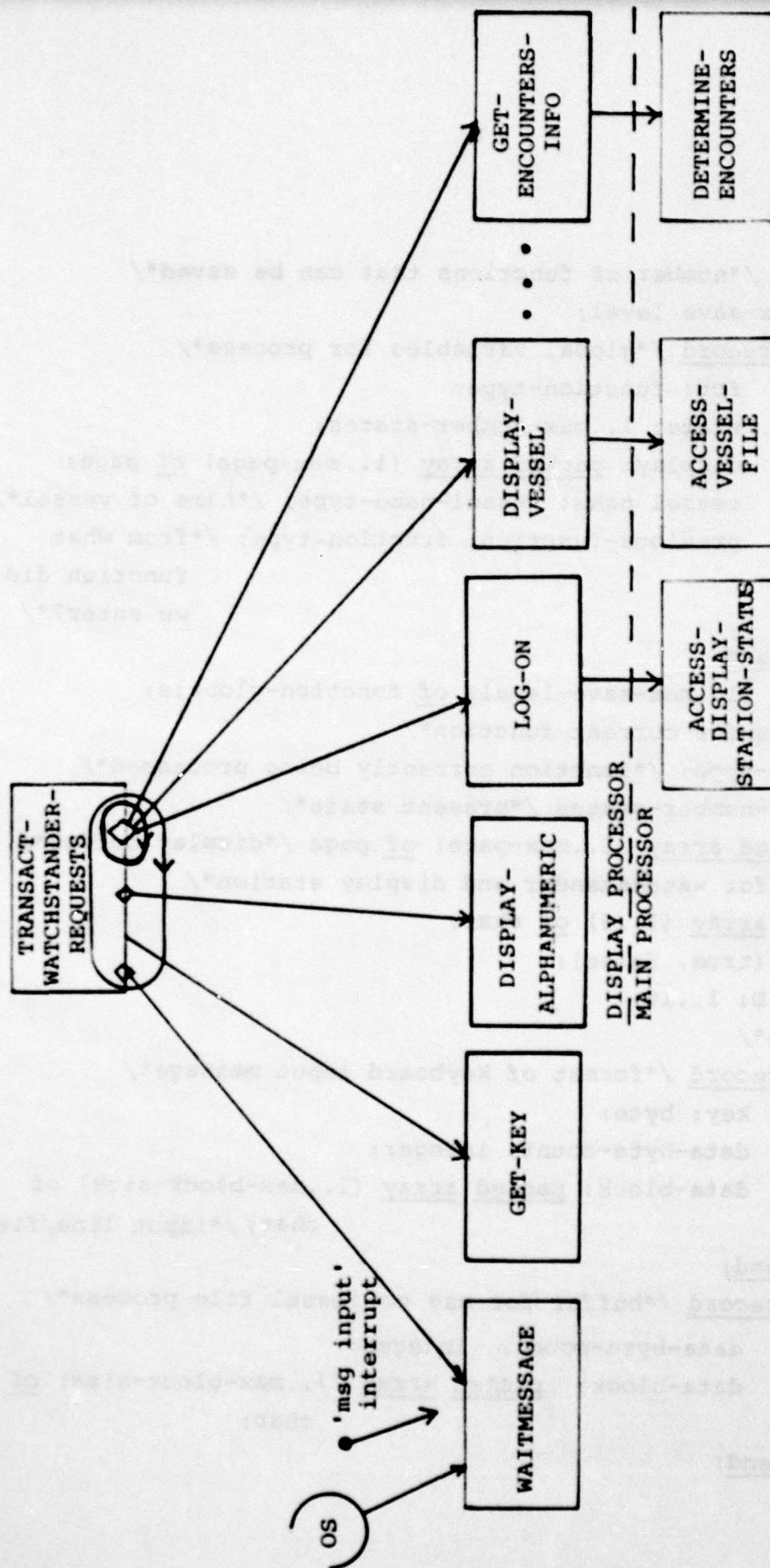


Figure 4-1. TRANSACTION-WATCHSTANDER-REQUESTS



Data Structures:

```
max-save-level=3; /*number of functions that can be saved*/
save-level: 0..max-save level;
function-globals=record /*global variables for process*/
    fct: function-type;
    state: 1..max-number-states;
    display: packed array (1..max-page) of page;
    vessel name: vessel-name-type; /*name of vessel*/
    previous-function: function-type; /*from what
                                         function did
                                         we enter?*/
    end;
save-stack: array (1..max-save-level) of function-globals;
/*global variables for current function*/
    fct: function-type; /*function currently being processed*/
    state: 1..max-number-states /*present state*/
    display: packed array (1..max-page) of page /*display buffer*/
/*identification for watchstander and display station*/
watchstander-ID: array (1..6) of char;
watchsupervisor: (true, false);
display-station-ID: 1..16;
/*message formats*/
input-msg-buf = record /*format of keyboard input message*/
    key: byte;
    data-byte-count: integer;
    data-block: packed array (1..max-block-size) of
                                         char; /*input line/field*/
    end;
output-msg-buf: record /*buffer for msg to vessel file process*/
    data-byte-count: integer;
    data-block: packed array (1..max-block-size) of
                                         char;
    end;
```

```
answer-buf: record /*buffer for answer from application process*/  
            data-byte-count: integer;  
            data-block: packed array (1..max-block-size) of  
                        char;  
  
            end;
```

Process TRANSACT-WATCHSTANDER-REQUESTS

```
begin state: = ready; watchstander-id = /; /*initialize variables*/
  while true /*infinite loop*/
    do
      if input-message-list is empty then wait for a message end;
      state: = initial;
      GET-KEY; /*get value of function key pressed*/
      if watchstander-id = ^ and key <> log-on then fct: =
        log-on else fct: = key;
      if (not fct in valid-function-set) and state <> ready
        then
          DISPLAY-ERROR ('INVALID FUNCTION');
          state: = ready;
        else
          while state <> ready;
            do
              case fct of
/*vessel functions*/      log-on: LOG-ON;
                           log-off: LOG-OFF;
                           display-vessel: DISPLAY-VESSEL;
                           enter-vessel: ENTER-VESSEL;
                           modify-vessel: MODIFY-VESSEL;
                           delete-vessel: DELETE-VESSEL;
/*passage functions*/    identify-vessel: IDENTIFY-VESSEL;
                           display-passage: DISPLAY-PASSAGE;
                           enter-passage: ENTER-PASSAGE;
                           modify-passage: MODIFY-PASSAGE;
                           update-vessel-position: UPDATE-VESSEL-
                                                    POSITION;
                           enter-communication: ENTER-COMMUNICATIONS;

                           change-status: CHANGE-STATUS;
```



```

/*data base info functions*/  local-traffic: GET-LOCAL-TRAFFIC;
                               environ-info: GET-ENVIRON-INFO;
                               search-on-key:  SEARCH-ON-KEY;
                               display-notice: DISPLAY-NOTICE;

/*update environ info*/      enter-man-env: ENTER-MAN-ENV-INFO;
                               modify-man-env: MODIFY-MAN-ENV-INFO;
                               delete-man-env: DELETE-MAN-ENV-INFO;
                               enter-forecast: ENTER-FORECAST;
                               modify-forecast: MODIFY-FORECAST;
                               delete-forecast: DELETE-FORECAST;

/*update notice*/           enter-notice: ENTER-NOTICE;
                               modify-notice: MODIFY-NOTICE;
                               delete-notice: DELETE-NOTICE;

/*update temporary route    enter-route-segment: ENTER-TEMP-
segment*/                   ROUTE-SEGMENT;

                               modify-route-segment:  MODIFY-TEMP-
                               ROUTE-SEGMENT;

                               delete-route-segment:  DELETE-TEMP-
                               ROUTE-SEGMENT;

/*alert response*/          alert-response: ALERT-RESPONSE;
/*demand calculations*/     encounters: GET-ENCOUNTERS-INFO;
                               relative-position: GET-RELATIVE-
                               POSITION-INFO;

                               cpa: GET-CPA-INFO;

/*route/schedule*/         route-schedule: ROUTE-SCHEDULE;
/*acknowledge*/            acknowledge: ACKNOWLEDGE;

/*simulations*/            record-scenario: RECORD-SCENARIO;
                               modify-scenario: MODIFY-SCENARIO;
                               delete-scenario: DELETE-SCENARIO;
                               initialize-scenario: INIT-SCENARIO;
                               start-scenario:  START-SCENARIO;

                               end /*case fct*/;
                               end /*while state < > ready*/;
                               end /*if not fct*/;
                               end /*while true*/;
                               end /*begin TRANSACT-WATCHSTANDER-REQUESTS*/

```

Procedure LOG-ON

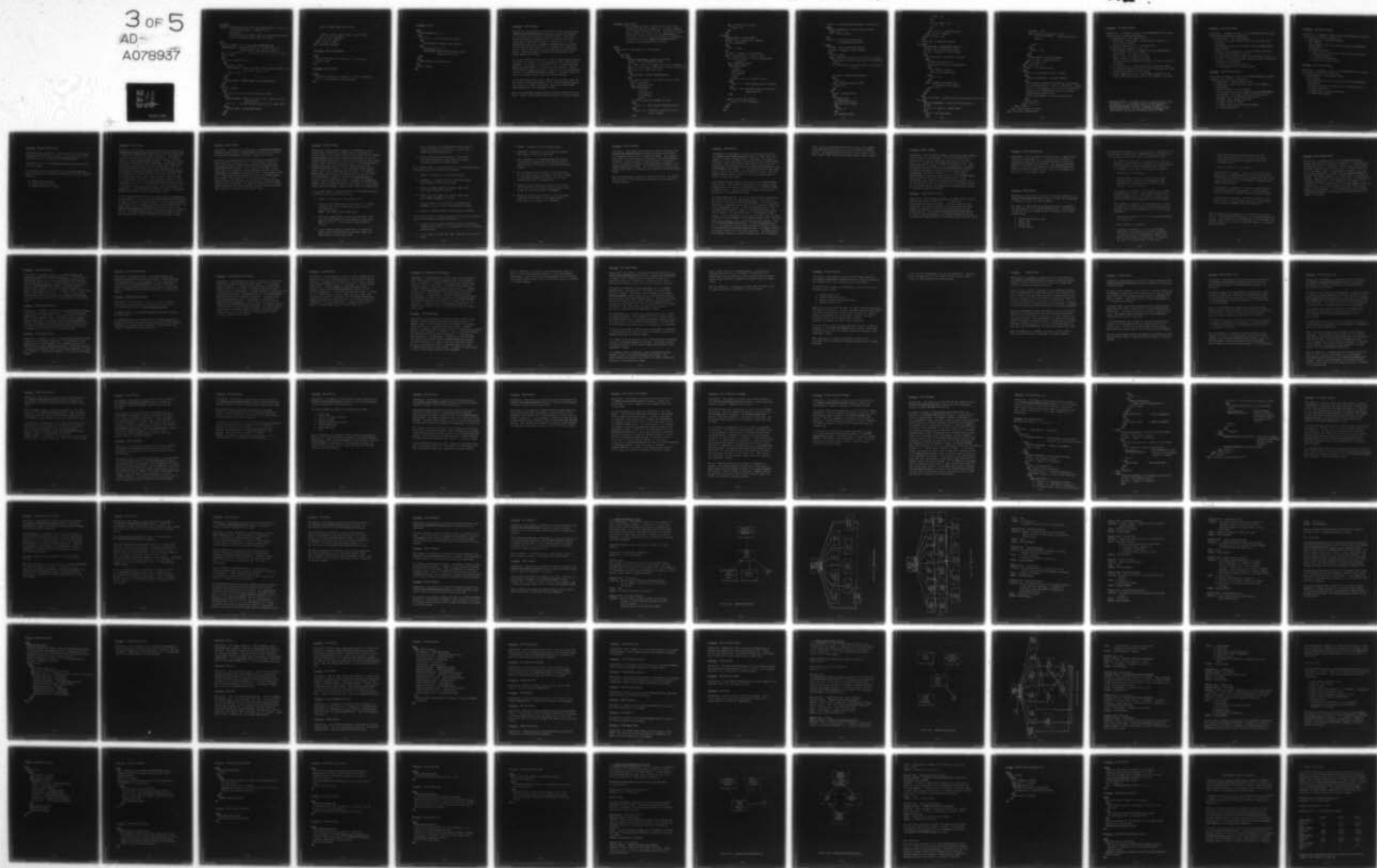
```
begin
  if watchstander-ID = ^, /*if not set*/
  then
    DISPLAY-ALPHANUMERIC ('ENTER ID');
    GET-KEY; /*get input*/
    if key <> carriage-return /*can only enter ID here*/
                                /*all other responses are invalid*/
    then
      DISPLAY-ERROR ('INVALID RESPONSE');
    else /*validate ID syntax and check DISPLAY-STATION-STATUS-TABLE
          to see if ID in use or if supervisor ID and supervisor
          already logged on system*/
      VALIDATE-ID (↑input-msg-buf; status);
      case status
        invalid: DISPLAY-ERROR ('INVALID ID');
        in-use: DISPLAY-ERROR ('ID ALREADY IN USE');
        supervisor-in-use: DISPLAY-ERROR ('SUPERVISOR ALREADY
                                         LOGGED ON');
        valid-supervisor-id: watchstander-id:= input-msg-buf.data-
                               block (1..6)
                               /*set 6-character ID*/;
                               watchsupervisor:= true;
        valid-watchstander-id: watchstander-id:=
                               input-msg-buf.data-block (1..6);
                               watchsupervisor:= false;
      end; /*case*/
    end; /*if key <> carriage-return*/
  else DISPLAY-ERROR ('ALREADY LOGGED ON');
  end; /*if watchstander-id*/
  state:= ready;
end; /*procedure LOG-ON*/
```

AD-A078 937

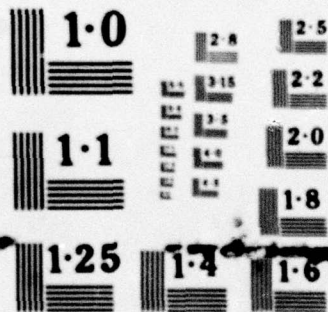
INTERNATIONAL COMPUTING CO BETHESDA MD  
VESSEL TRAFFIC SERVICES PROCESSING/DISPLAY SUBSYSTEM SOFTWARE R--ETC(U)  
SEP 79 C C HENSON , R S GRAHAM , B A MCINTOSH DOT-CG-81-78-1833  
USCG -D-73-79 NL

UNCLASSIFIED

3 OF 5  
AD-  
A078937







NATIONAL BUREAU OF STANDARDS  
MICROCOPY RESOLUTION TEST CHART

Procedure

GET-KEY /\*get keyboard input from input message buffer (first byte).  
Perform SAVE, RECALL, or ESCAPE if key so  
indicates.\*/  
/\*should test fct and state upon return from this procedure  
in case changed by RECALL, SAVE, or ESCAPE\*/

begin

if input message list is empty then WAITMESSAGE end;  
TRANSFERMESSAGE (sending-process, ID, ptr-msg-buffer, ptr-  
answer-buffer); /\*get keyboard input\*/  
key:= ptr-msg-buffer; /\*get key code from 1st byte of buffer\*/  
if key = SAVE  
then  
    if level < max-save-level  
    then  
        if state <> ready  
        SAVE-GLOBALS; /\*saves fct, state, level, display buffer, etc.,  
                          then level:= level + 1\*/  
        state:= ready  
    end;  
    else  
        DISPLAY-ERROR ('MAXIMUM NUMBER OF FUNCTIONS SAVED')  
    end;  
else  
    if key = RECALL  
    then  
        if level > 0 /\*if at least one function saved\*/  
        then  
            RECALL-GLOBALS; /\*sets fct, state, level, display to last  
                              saved values\*/  
            key:= fct /\*in case GET-KEY called while in ready state\*/  
        else  
            DISPLAY-ERROR ('NO FUNCTIONS SAVED')  
        end;  
    else

```

    if key = ESCAPE and state <> ready

        state: = ready; /*put system in ready state*/
    end; /*if key = ESCAPE*/
    end; /*if key = RECALL*/
    end; /*if key = SAVE*/
end; /*procedure GET-KEY*/

```

Procedure DISPLAY-ALPHANUMERIC

```

begin
    Output alphanumeric data specified to alphanumeric
    display screen;
end;

```

Procedure DISPLAY-ERROR

```

begin
    Output error message in response to illegal watchstander
    input to the alphanumeric display screen;
end;

```



## Procedure LOG-OFF

begin

if watchstander-id < > ^

then

if level > 0 /\*if any functions saved\*/

then

DISPLAY-ERROR ('RECALL SAVED FUNCTION');

else

watchstander-id: = ^ ;

DELETE-FROM- DISPLAY-STATION-STATUS-TABLE

(display-station-ID);

end;

else

DISPLAY-ERROR ('NOT LOGGED ON');

end;

state: = ready;

end;

## Procedure    DISPLAY-VESSEL

Description: This module is called from the main program of the TRANSACT-WATCHSTANDER-REQUESTS process to display the contents of the vessel file record of a specified vessel. Upon entry, the module displays a numbered list of the ways in which the vessel may be identified (see Part I, Section 6.2.2). The watchstander types the list number corresponding to the identification method which he desires. The module then prompts him for the type of identification input required for the specified method. The module sends a Get-Vessel-Record message to the ACCESS-VESSEL-FILE process.

If the watchstander specified the 3-letter wildcard identification, the answer will contain a list of qualified vessel names and disc addresses. The module will display the list of vessel names and prompt the watchstander to select one of the vessels. A Read-Vessel-Record message with the disc address of the selected vessel is sent to the ACCESS-VESSEL-FILE process. The answer contains a copy of the vessel record, which this module displays.

If the watchstander specified an identification method other than the 3-letter wildcard, the answer to the Get-Vessel-Record message will contain the vessel record data, which the DISPLAY-VESSEL module displays on the alphanumeric screen.

When the watchstander presses ENTER, the display station state is set to ready and the module returns control to the main program.



Procedure ENTER-VESSEL

/\*This procedure prompts, inputs, and verifies vessel information from the watchstander. It sends a message to the ACCESS-VESSEL-FILE process, containing the vessel record (fields not filled in by the watchstander are left blank). A list of the data items with unreasonable values is sent to the MANAGE-ACTION-REQUIRED-LIST process at the watchsupervisor station\*/

begin

while state <> ready and fct = enter-vessel

do

case state

2A: begin

DISPLAY-ALPHANUMERIC (vessel-record-form);

GET-KEY; /\*get watchstander input\*/

if vr.name = blank /\*if vessel name not yet entered\*/

and fct = enter-vessel /\*and still in same function and\*/

and state = 2A /\*same state then\*/

then

DISPLAY-ERROR ('VESSEL NAME REQUIRED');

else

if vr.name contains a contiguous string of alphanumeric characters

then /\*reasonable name\*/

GET-VESSEL-RECORD

(vr.name;

vessel-record;

disc-address;

found)

if found = true /\*if already in file\*/

then

state: = 3 /\*then display existing record \*/

else

state: = 4A /\*otherwise continue filling in vessel record\*/

end;



```

        else /*invalid name syntax*/
            state: = 2B
        end;
    end;
2B: begin
    DISPLAY-ERROR ('ILLEGAL NAME');
    GET-KEY; /*get watchstander response*/
    if key = move-cursor
    then
        state: = ready
    end;
    if state = 2B and fct = enter-vessel
    then /*watchstander has re-typed name*/
        if vr.name contains a string of alphanumeric
            characters
        then /*reasonable name*/
            GET-VESSEL-RECORD
                (vr.name;
                vessel-record;
                disc-address;
                found)
            if found = true /*if already in file*/
            then
                state: = 3 /*then display existing record*/
            else
                state: = 4A /*otherwise continue filling in
                    vessel record*/
            end;
        end;
    else /*invalid name syntax*/
        state: = 2B /*so do again*/
    end;
    end;
end;

```

```

/*state 3 - Requested Record Already In Vessel File*/
3 : begin
    DISPLAY-ALPHANUMERIC (existing vessel record);
    state: = 3;
    fct: = modify-vessel;

end;
/*State 4A - Partially Filled Record
- No Unreasonable Entries*/
4A: begin
    GET-KEY; /*get watchstander input*/
    if state = 4A and fct = enter-vessel
    then
        if key = ENTER
        then /*Send Vessel Record to be Entered in File*/
            Format Vessel Record; Create List of Unreasonable
            Entries;
            ADD-VESSEL-RECORD (↑vessel-record; disc-address;
                               status)

            if status = vessel-record-entered
            then
                state: = 5
            else /*existing vessel record*/
                state: = 4C
            end;
        else
            if key = carriage-return
            then
                validate entry
                if unreasonable entry
                and name field changed
                then
                    state: = 3
                end;
                if reasonable entry

```



```

        state: = 4A
    end;

    if unreasonable entry
        state: = 4B
    end;

    end; /*if key - CARRIAGE RETURN*/
    end; /*if key - ENTER*/
    end; /*if state - 4A*/
end;
/*State 4B - Unreasonable Entry*/
4B: begin
    DISPLAY-ERROR ('UNREASONABLE ENTRY');
    GET-KEY; /*get watchstander input*/
    if state = 4B and fct = enter-vessel
    then
        if unreasonable entry field is changed
        then
            validate new entry value
            if new value of entry is unreasonable
            then
                flash field on display
                /*and stay in same state*/
            else
                state: = 4A
            end;
        else /*criteria overridden and
            unreasonable value remains*/
            state: = 6A
        end;
    end
end;
/*Ask W/S whether to substitute for existing record in file*/
4C: begin
    DISPLAY-ALPHANUMERIC (vessel-record-exists-form);
    GET-KEY;
    if state = 4C and fct = enter-vessel
    then
        if key = carriage-return
        then

```



```

    if input = 'yes'
    then /*write over existing record*/
        WRITE-VESSEL-RECORD (↑ vessel-record; disc
            address)

        State:= 5
    else
        State:= ready
    end;
    else
        DISPLAY-ERROR ('INVALID RESPONSE')
    end; /*if key = carriage-return*/
    end; /*if state = 4C*/
end; /*state 4C*/
/*State 5 - All Data Entered*/
5 : begin
    if List of Unreasonable Entries is Empty
    then
        DISPLAY-ALPHANUMERIC ('ALL DATA ENTERED')
    else
        Send message to MANAGE-ACTION-REQUIRED-LIST process
        of the watchsupervisor's station, containing 1) vessel-
        record, 2) disc-address, and 3) list of unreasonable
        entries; Place unreasonable entries in all-data-
        entered-except display form;
        DISPLAY-ALPHANUMERIC (all-data-entered-except)
    end;
    State:= ready;
    end; /*state 5 */
end; /*case*/
end; /*while state <> ready*/
end; /*procedure ENTER-VESSEL*/

```

\* Procedure GET-VESSEL-RECORD

Description: A message is sent to the ACCESS-VESSEL-FILE process containing the following information:

- 1) msg-function: = get-vessel-record
- 2) index-type - vessel name, 3-letter-name-wildcard, call sign or Lloyd's number
- 3) value of index key (e.g., the vessel name)
- 4) modification-intended: = yes or no

The procedure waits for an answer from the ACCESS-VESSEL-FILE process. The answer contains:

- 1) found = true if a record was located  
found = false if no record was found  
If found then the following information is also contained in the answer:
- 2) If index-type is 3-letter-name-wildcard, then the number of qualified records found, and a list containing the vessel name and disc address of each vessel.
- 3) If the index-type is not 3-letter-name-wildcard then the single vessel record copy and the disc address is returned.

\* Asterisks indicate procedures which are called by higher level procedures (i.e., a procedure without an asterisk which is called directly from the main program of the TRANSACT-WATCHSTANDER-REQUESTS process) to request access to a given file or record type. See the higher level procedures for references to the specific procedures called by them.



\* Procedure ADD-VESSEL-RECORD

Description: A message is sent to the ACCESS-VESSEL-FILE process containing the following information:

- 1) msg-function: = add-vessel-record
- 2) a copy of the vessel record with the vessel name field filled-in.

The procedure waits for an answer from the ACCESS-VESSEL-FILE process.

- 1) status, which indicates one of the following conditions
  - a) vessel record added to the file
  - b) vessel record (of same name) already exists in the file
- 2) If there is an existing vessel record then the following are contained in the message:
  - a) a copy of the existing record
  - b) disc address of the existing record

\* Procedure DELETE-VESSEL-RECORD

Description: A message is sent to the ACCESS-VESSEL-FILE process containing the following information:

- 1) msg-function: = delete-vessel-record
- 2) index-type = vessel name, call sign, Lloyd's number, or disc address
- 3) value of index key or disc address

The procedure waits for an answer from the ACCESS-VESSEL-FILE process. The answer contains the status which indicates one of the following conditions:

- a) vessel record deleted from file
- b) vessel record not found
- c) vessel record currently being accessed
- d) invalid index value



\* Procedure READ-VESSEL-RECORD

Description: A message is sent to the ACCESS-VESSEL-FILE process containing the following information:

- 1) msg-function: = read-vessel-record
- 2) disc address

The procedure waits for an answer from the ACCESS-VESSEL-FILE process.

The answer contains:

- 1) status which indicates one of the following conditions:
  - a) read successful
  - b) invalid disc address
- 2) a copy of the vessel record

\* Procedure WRITE-VESSEL-RECORD

Description: A message is sent to the ACCESS-VESSEL-FILE process containing the following information:

- 1) msg-function: = write-vessel-record
- 2) disc address

The procedure waits for an answer from the ACCESS-VESSEL-FILE process.

The answer contains the status which indicates one of the following conditions:

- a) vessel record written successfully
- b) invalid disc address

\* Procedure RELEASE-VESSEL-RECORD

Description: This procedure is called to unlock a vessel record which has been accessed with intent to write, but has not been unlocked by a call to procedure WRITE-VESSEL-RECORD.

A message is sent to the ACCESS-VESSEL-FILE process containing the vessel name.

The procedure waits for an answer from the ACCESS-VESSEL-FILE process containing the status which indicates one of the following conditions:

- a) vessel record released
- b) vessel record not locked
- c) vessel record does not exist



Procedure    MODIFY-VESSEL

Description: This module is called from the main program of the TRANSACT-WATCHSTANDER-REQUESTS process to allow the watchstander to change the contents of a vessel file record. The vessel is identified and the record accessed and displayed in the same manner as in the DISPLAY-VESSEL procedure. The module now moves the cursor as specified by the watchstander, and changes values in fields to new values typed by the watchstander. Each new value is checked for correct syntax and reasonableness. If it is outside the range of reasonable values then the watchstander is notified on the display screen of the error condition. If the unreasonable value is not corrected then an entry is added to an Unreasonable-Value-List, which contains 1) the old value, 2) the new value, and 3) the reasonable range of values. If an unreasonable value is not corrected then the value of the field is not changed in the vessel record buffer. When the ENTER key is pressed then a Write-Vessel-Record message is sent to the ACCESS-VESSEL-FILE process with the updated vessel record.

If the Unreasonable-Value-List is not empty then a Get-Supervisor-Station message is sent to the ACCESS-DISPLAY-STATION-STATUS process. The answer contains the display station ID of the watch supervisor. An Add-Entry-to-List message containing a display message text, old vessel record, and the Unreasonable-Value-List is sent to the MANAGE-ACTION-REQUIRED-LIST process at the watch supervisor display station. The MODIFY-VESSEL module then returns control to the main program, after setting the display station state to ready.



Procedure DELETE-VESSEL

Description: This module is called from the TRANSACT-WATCHSTANDER-REQUESTS process when the 'DELETE-VESSEL' function key is pressed. It allows the watchstander to delete a vessel file record.

Upon entry, the module prompts the watchstander to identify the vessel (see Part I, Section 6.2.2). Once the vessel has been identified the module sends a Get-Vessel-Record message to the ACCESS-VESSEL-FILE process. If the record is contained in the answer then it is displayed on the alphanumeric screen, and the watchstander is prompted to enter YES or NO. If the answer is 'YES' then a Delete-Vessel-Record message is sent to the ACCESS-VESSEL-FILE process. The answer may be 1 of the following: 1) the vessel record is now deleted or 2) the record is currently being used and cannot be deleted. The answer is displayed on the alphanumeric screen and the module returns control to the main program after setting the display station state to ready.

## Procedure IDENTIFY-VESSEL

Description: This procedure enables a watchstander to link a tracked vessel displayed on the map screen to the existing passage record. It is called from the main program of the TRANSACT-WATCHSTANDER-REQUESTS process when the watchstander presses the IDENTIFY-VESSEL function key. The module requests and inputs vessel identification (see Part I, Section 6.2.2). A Get-Passage-Record message is then sent with the vessel identification to the ACCESS-PASSAGE-FILE process. If the specified passage record exists, then it will be returned in the answer. The IDENTIFY-VESSEL module then displays the passage record data and prompts the watchstander to "hook" the vessel on the map display. A Get-Identification message is sent to the MANAGE-MAP-DISPLAY process. When the HOOK function key is pressed, the MANAGE-MAP-DISPLAY returns an answer containing the internal ID of the vessel hooked, and the (external) vessel ID code (blank if unidentified).

If the hooked vessel is unidentified and the watchstander presses the ENTER key, then the following is done:

- 1) HOOKID: = internal ID of the hooked vessel.
- 2) Search the ID-Tracker-Table (see Section 3.1.1) entries 0 through 3999 (IDs for identified vessels) for the first empty entry.  
NEWID: = the number of the empty entry.
- 3) From entry number HOOKID in the ID-Tracker-Table (IDT) move the radar number and track number to entry number NEWID in the IDT. Clear the entry number HOOKID in the IDT.
- 4) In the Tracker-ID-Table (see Section 3.1.1) move the NEWID to the entry specified by the radar number and track number in the previous step.



- 5) Send a message to the MANAGE-MAP-DISPLAY process to update the internal ID and vessel ID code of the Vessel-Information-Table entry.
- 6) Send a Clear-Hook-and-Set message to the MANAGE-MAP-DISPLAY process to remove the circle around the hooked vessel on the map display.

If the hooked vessel is unidentified and the watchstander presses the LINK key, then the following is performed:

- 1) HOOKID: = internal ID of the hooked vessel.
- 2) PASSAGEID: = internal ID of the identified vessel whose passage record has been read.
- 3) Obtain the radar number and tracker number from entry number HOOKID of the IDT.
- 4) Obtain the radar number and tracker number from entry number PASSAGEID of the IDT.
- 5) Set the pointer in the TID entry corresponding to PASSAGEID to the TID entry corresponding to HOOKID.
- 6) Delete the Vessel-Information-Table entry for HOOKID.

If the hooked vessel is already identified, then do the following when the watchstander presses the ENTER key.

- 1) Prompt the watchstander, asking him whether he wishes to reidentify the vessel (i.e., reassociate it with a different passage record).
- 2) If the answer is other than 'YES', then skip the following steps.



- 3) HOOKID: = internal ID of the hooked vessel.
- 4) PASSAGEID: = internal ID of the identified vessel whose passage record has been read.
- 5) Send a message to the ACCESS-PASSAGE-FILE process requesting the passage with the external vessel ID code returned in the answer to the Get-Identification message.
- 6) For the passage record obtained in the above step, set the status field to 'imminent' and send a message to the ACCESS-PASSAGE-FILE process to write the updated record into the passage file.
- 7) Change the Vessel-Information-Table entry, setting the internal ID, and external vessel ID code to the values in the passage record of PASSAGEID.
- 8) Change the IDT and the TID table to have the radar number and track number of the hooked vessel to correspond to the internal ID = PASSAGEID.

## Procedure    DISPLAY-PASSAGE

Description: This module is called from the main program of the TRANSACT-WATCHSTANDER-REQUESTS process to display the contents of the passage file record of a specified vessel. The vessel is identified in the same manner as described in the DISPLAY-VESSEL procedure. Once the vessel has been identified, the passage record is obtained by sending a Get-Passage-Record message to the ACCESS-PASSAGE-FILE process. The answer contains a copy of the passage record which is formatted and displayed on the alphanumeric screen.

When the watchstander presses the ENTER function key, the display station state is set to ready and the module returns control to the main program.



#### Procedure ENTER-PASSAGE

Description: This module is called from the main program of the TRANSACT-WATCHSTANDER-REQUESTS process to enable the watchstander to create a passage record. Upon entry, the vessel is identified in the same manner as described for the DISPLAY-VESSEL procedure. Upon receiving the vessel identification from the watchstander, this module sends a Get-Vessel-Record message to the ACCESS-VESSEL-FILE process. If the answer specifies that there is not a vessel file record for the identified vessel, then the ENTER-VESSEL procedure is called to allow the watchstander to create a vessel file record.

A Get-Passage-Record message is now sent to the ACCESS-PASSAGE-FILE process. If the answer specifies that the passage record already exists then control is transferred to the MODIFY-PASSAGE procedure; else, a 'fill-in-the-blank' passage record form is displayed on the alphanumeric screen.

The watchstander may now fill in the blank fields by positioning the cursor and typing the appropriate values. Each value is checked for correct syntax and reasonableness. If it is outside the range of reasonable values then an error message is displayed on the alphanumeric screen. If the unreasonable value is not corrected then an entry is added to an Unreasonable-Value-List, which contains 1) the new value, and 2) the reasonable range of values. If an unreasonable value is not corrected, then the value of the field is not changed in the passage record buffer. When the ENTER key is pressed, then a Write-Passage-Record message is sent to the ACCESS-PASSAGE-FILE process. If the Unreasonable-Value-List is not empty then a Get-Supervisor-Station message is sent to the ACCESS-DISPLAY-STATION-STATUS process. The answer contains the display station ID of the watch supervisor. An Add-Entry-to-List message containing a display message text, the old passage



record, and the Unreasonable-Value-List is sent to the MANAGE-ACTION-REQUIRED-LIST process at the watch supervisor display station. The ENTER-PASSAGE module then returns control to the main program, after setting the display station state to ready.

#### Procedure MODIFY-PASSAGE

Description: This procedure requests and receives vessel identification from the watchstander. If the watchstander opts for identification with the map 'hook' function, this program waits for a message from the MANAGE-MAP-DISPLAY process. The message contains the vessel ID code (4 characters). It obtains the passage record by calling the GET-PASSAGE-RECORD procedure. It displays the passage record and inputs changes from the watchstander (see Part I, Section 8.2). It calls WRITE-PASSAGE-RECORD to send the updated record to the ACCESS-PASSAGE-FILE process for writing into the passage file. If any unreasonable data items are entered by the watchstander and are not corrected then they are not entered into the passage record, but are instead sent to the MANAGE-ACTION-REQUIRED-LIST process at the watch supervisor's station.

#### Procedure UPDATE-VESSEL-POSITION

Description: This procedure performs the Update Vessel Position function (see Part I, Section 8.5). It prompts the watchstander to identify a vessel. Once the vessel has been identified, the vessel information is displayed and the watchstander is prompted to enter the information concerning the vessel position. The procedure MODIFY-REPORTING-POINT-RECORD is called to send a message to the ACCESS-PASSAGE-FILE process to update the vessel position in the proper reporting point record.



#### Procedure ENTER-COMMUNICATIONS

Description: This procedure is called to enter a communications record into the passage file. It prompts the watchstander to identify the vessel and calls procedure GET-KEY to input the communications summary and date/time. The procedure ADD-COMM-RECORD is called to send a message to the ACCESS-PASSAGE-FILE process to write the communications record in the passage file and link it to the passage record.

#### Procedure CHANGE-STATUS

Description: This module is called from the main program of the TRANSACT-WATCHSTANDER-REQUESTS process to enable the watchstander to change the status of a vessel passage.

The vessel is identified and the passage record is accessed in the same manner as in the MODIFY-PASSAGE module. The following information from the passage record is displayed on the alphanumeric screen:

1. Vessel identification code
2. Vessel name
3. Vessel type
4. Present status

The module then displays a list of the possible statuses for the watchstander to choose from. Depending on the old status and the new status, the following data is requested and input:

- . From imminent to underway - The subsystem will ask the operator to either identify the vessel on the map display (if the vessel is within the coverage area of level 4 or 5 sensors), or to enter the following information:
  - Designation of next waypoint
  - Average speed of advance to next waypoint, OR the estimated time of arrival at next waypoint (the subsystem will calculate and enter the value the operator chose not to enter).
- . From underway to imminent - The subsystem will transfer the passage data to the "vessel passage history" and "traffic summary" files. All waypoint information will then be deleted from the passage file for that vessel.
- . From underway to anchored - The subsystem first invokes the underway to imminent functions described above then displays a blank form requesting entry of the following information:
  - Anchorage Designation (if it is a standard predefined anchorage point)
  - Swing radius of the mooring
  - Location of anchorage (if it is a non-standard anchorage) in Latitude and Longitude or described in plain English using references to nearby landmarks, and if there is a map display, the position may also be identified using the map cursor.



- Date/time anchorage was established (This entry will automatically contain the current date/time which may be changed by the operator if this was a delayed report).
- Date/time expected to get underway
- . From anchored to underway - Records the anchorage information into the vessel passage history file then deletes the anchorage information from the passage file data, then invokes the functions for the imminent to underway status change described above.
- . From imminent to anchored - Provides the same functions as the status change from underway to anchored except that the underway to imminent functions are not involved.
- . From anchored to imminent - Records the anchorage information into the vessel passage history file then deletes the anchorage information from the passage file.

When the ENTER key is pressed, the new communication record is sent in a Write-Passage-Record message to the ACCESS-PASSAGE-FILE process. Upon receiving an acknowledgement for the message, the display station status is set to ready and control is returned to the main program.

\* Procedure GET-PASSAGE-RECORD

Description: A message is sent to the ACCESS-PASSAGE-FILE process containing the following information: 1) msg-function:= get-passage-record; 2) index-type - vessel name, 3-letter-name-wildcard, vessel ID code, or pilot name; 3) value of index key (e.g., the vessel name); 4) modification-intended:= yes or no. The procedure waits for an answer from the ACCESS-PASSAGE-FILE process. The answer contains: 1) found = true if a record was located; found = false if no record was found. If found then the following information is also contained in the answer: 2) If index-type is 3-letter-name-wildcard, then the number of qualified records found, and a list containing the vessel name and disc address of each passage; 3) If the index-type is not 3-letter-name-wildcard then the single passage record copy and the disc address is returned.



\* Procedure ADD-PASSAGE-RECORD

Description: A message is sent to the ACCESS-PASSAGE-FILE process containing the following information: 1) msg-function:= add-passage-record. 2) a copy of the passage record with the vessel name field filled-in. The procedure waits for an answer from the ACCESS-PASSAGE-FILE process. The answer contains: 1) status, which indicates one of the following conditions, a) passage record added to the file, b) passage record (of same vessel name) already exists in the file. 2) If there is an existing passage record then the following are contained in the message: a) a copy of the existing record, b) disc address of the existing record.

\* Procedure DELETE-PASSAGE-RECORD

Description: A message is sent to the ACCESS-PASSAGE-FILE process containing the following information: 1) msg-function:=delete-passage-record, 2) index-type - vessel name vessel ID code, pilot name, or disc address, 3) value of index key or disc address. The procedure waits for an answer from the ACCESS-PASSAGE-FILE process. The answer contains the status which indicates one of the following conditions: a) passage record deleted from file, b) passage record not found, c) invalid index value.

\* Procedure READ-PASSAGE-RECORD

Description: A message is sent to the ACCESS-PASSAGE-FILE process containing the following information: 1) msg-function:= read-passage-record, 2) disc address. The procedure waits for an answer from the ACCESS-PASSAGE-FILE process. The answer contains: 1) Status which indicates one of the following conditions; a) read was successful, b) invalid disc address. 2) A copy of the passage record.

\* Procedure WRITE-PASSAGE-RECORD

Description: A message is sent to the ACCESS-PASSAGE-FILE process containing the following information: 1) msg-function:= write-passage-record; 2) disc address. The procedure waits for an answer from the ACCESS-PASSAGE-FILE process. The answer contains the status which indicates one of the following conditions: a) passage record written successfully; b) invalid disc address.

\* Procedure RELEASE-PASSAGE-RECORD

Description: This procedure is called to unlock a passage record which has been accessed with intent to write, but has not been unlocked by a call to procedure WRITE-PASSAGE-RECORD.

A message is sent to the ACCESS-PASSAGE-FILE process containing the vessel name.

The procedure waits for an answer from the ACCESS-PASSAGE-FILE process containing the status which indicates one of the following conditions: a) passage record released; b) passage record not locked; c) passage record does not exist.



\*Procedure GET-REPORTING-POINT-RECORD

Description: This procedure is called to obtain a copy and disc address of an existing reporting point record. The following input is required: 1) passage record index type = vessel name, vessel ID code, or pilot name; 2) passage record index value; 3) reporting point designation; 4) memory address of a buffer for the reporting point record. A message is sent to the ACCESS-PASSAGE-FILE process with the following information: 1) msg-function= get-reporting-point-record; 2) index-type; 3) index-value; 4) reporting point designation. The procedure then waits for an answer from the ACCESS-PASSAGE-FILE process. The answer contains: 1) status = one of the following indications, a) reporting point record found, b) passage record not found, c) reporting point record not found; 2) disc address of reporting point record; 3) copy of reporting point record.

\* Procedure ADD-COMM-RECORD

Description: This procedure is called to write a communications record and link it to a passage record. The following input is required: 1) passage record index type = vessel name, 3-letter-name-wildcard, vessel ID code, or pilot name. 2) passage record index value, 3) memory address of communications record. A message is sent to the ACCESS-PASSAGE-FILE process with the following information: 1) msg-function=add-comm-record, 2) index-type, 3) index-value, 4) copy of communications record. The procedure then waits for an answer from the ACCESS-PASSAGE-FILE process containing the status with one of the following possible indications: 1) comm record added, 2) passage record doesn't exist, 3) invalid index type, 4) invalid index value.



\*Procedure ADD-REPORTING-POINT-RECORD

Description: This procedure is called to write a new reporting point record and link it to a passage record. The following input is required: 1) Passage record index type = vessel name, vessel ID code, or pilot name. 2) Passage record index value. 3) Memory address of copy of reporting point record. A message is sent to the ACCESS-PASSAGE-FILE process with the following information: 1) msg-function = add-reporting-point-record, 2) index-type, 3) index-value, 4) copy of reporting point record. The procedure then waits for an answer from the ACCESS-PASSAGE-FILE process containing the status set to indicate one of the following: 1) reporting point record added successfully, 2) passage record doesn't exist, 3) invalid index type, 4) invalid index value.

\*Procedure GET-COMM-RECORDS

Description: This procedure is called to obtain a copy of disc address of every existing communications record linked to a passage record. This following input is required: 1) passage record index type, 2) passage record index value, 3) memory address of buffer in which to place the list of communications records and their disc addresses. A message is sent to the ACCESS-PASSAGE-FILE process with the following information: 1) msg-function= get-comm-records, 2) index-type, 3) index-value. The procedure then waits for an answer from the ACCESS-PASSAGE-FILE process. the answer contains: 1) status = one of the following indications, a) comm record(s) found, b) passage record not found, c) passage record has no comm records. 2) number of comm records, 3) list of comm records and their disc addresses.

When the ENTER key is pressed the new communication record is sent in a Write-Passage-Record message to the ACCESS-PASSAGE-FILE process. Upon receiving an acknowledgement for the message the display station status is set to ready and control is returned to the main program.



## Procedure GET-LOCAL-TRAFFIC

Description: This module is called from the main program of the TRANSACT-WATCHSTANDER-REQUESTS process to enable the watchstander to obtain a list of vessels within a specified radius of a specified vessel, and to display the contents of the vessel record and/or passage record of any of the vessels on the list.

The vessel is identified in the same manner as in the MODIFY-PASSAGE module. The position of the vessel is obtained from the Position-Table. The module sends a message to the ACCESS-SYSTEM-PARAMETERS process requesting the Local-Traffic-Default-Radius and the Local-Traffic-Maximum-Radius. The module searches the Position-Table for the vessels within the default radius, creating a Local-Traffic-List. The Local-Traffic-List is sorted by distance. The vessel name and type for each vessel on the list is displayed on the alphanumeric screen.

The watchstander may now press the carriage return key or ENTER to change the radius. He is prompted for the new radius, and a new Local-Traffic-List is created and displayed, unless the radius is greater than the Local-Traffic-Maximum-Radius; in which case, an error message is displayed and the watchstander is re-prompted.

If a list entry number is entered, the watchstander is requested to specify whether he wishes to display the vessel record or the passage record for the selected vessel.

If a vessel record is specified, then a Read-Vessel-Record message is sent to the ACCESS-VESSEL-FILE process. The answer contains the vessel record, which the module formats and displays on the alphanumeric screen.

If a passage record is specified, then a Read-Passage-Record message is sent to the ACCESS-PASSAGE-FILE process. The answer contains the vessel record, which the module formats and displays on the alphanumeric screen.

When a vessel record or a passage record is currently being displayed when the YES key is pressed, the Local-Traffic-Vessel-List is again displayed, and the watchstander may select another vessel on which to display the passage record or vessel record.

When the ESCAPE key is pressed, the display station state is set to ready and control is returned to the main program.



Procedure GET-ENVIRON-INFO

Description: This module is called from the main program of the TRANSACT-WATCHSTANDER-REQUESTS process to display information environmental data files requested by the watchstander.

The module first prompts the watchstander to select one of the following types of data:

1. Weather Sensor Data
2. Current/Tide Sensor Data
3. Manually Entered Environmental Data
4. Weather Forecast Data

Based on the type of data selected, the module requests and inputs the appropriate selection criteria. Weather sensor data and current/tide data are selected by sensor station designation. Manually entered environmental data is selected by sector, and within the sector from a list of report times. Weather forecasts are selected by sector and by date/time.

Once the selection data has been specified, the data is sent in a message to the ACCESS-ENVIRONMENTAL-DATA process. The answer contains the data, which the module formats and displays on the alphanumeric screen.

When sensor data is currently displayed, if the NO key is pressed then the previously displayed selection list is again displayed.

At any time the watchstander may press the ESCAPE key. Upon this event, the module will set the display station state to ready. Control is then returned to the main program.



Procedure

SEARCH-ON-KEY

Description: This module is called from the main program of the TRANSACT-WATCHSTANDER-REQUESTS process to enable the watchstander to display qualifying search output data items after providing the search parameters desired.

First, a list of files is displayed on the alphanumeric screen, and the watchstander is prompted to select a file from the list. Then, a list of the data items contained in each record of the file is displayed and the watchstander is prompted to select one or more of these fields to be output from the search. Then, a list of search keys is displayed and the watchstander is prompted to type the logical expressions specifying the search parameters.

When the watchstander presses the ENTER key to enter the search parameters, the SEARCH-ON-KEY process searches the specified file for each record whose values match the values specified for the fields selected. The watchstander is notified that the key search is underway and when the search is completed, the ACTION REQUIRED indicator light will turn on. The watchstander will then press the ACKNOWLEDGE key to display the output data items.

When the ESCAPE key is pressed, the display station state is set to ready and control is returned to the main program.

## Procedure    DISPLAY-NOTICE

Description: This module is called from the main program of the TRANSACT-WATCHSTANDER-REQUESTS process to display a record from the notices file.

The module first displays a list of notice types and requests the watchstander to select one of the types or all or the the types. After inputting the type selection, the module is requested to specify a sector. Valid input here is a sector designation or '\*' to specify all sectors.

The module sends a Get-Notices-Records message containing the specified notice type(s) and sector(s) to the ACCESS-NOTICES-FILE process. The answer contains the qualified notice records found. The module formats and displays on the alphanumeric screen the first notice in the list.

If the watchstander then presses the carriage return key, the module will get and display the next notice record in the notices list. If there are no more entries in the list, the module requests another sector to be input by the watchstander.

When the watchstander presses the ESCAPE key, the module sets the display station status to ready and returns control to the main program.



Procedure ENTER-MAN-ENV-INFO

Description: This module is called from the main program of the TRANSACT-WATCHSTANDER-REQUESTS process to enable the watchstander to enter an environmental data record in the data base.

The module requests the watchstander to select one of three methods to specify the location of the environmental data:  
1) type the latitude and longitude, 2) position the cursor on the map display and press the SET key, or 3) select from a list of landmarks.

When the watchstander has entered the location, the module asks whether he wishes to enter a range and bearing relative to the location. If so, the final position is computed using the range and bearing relative to the location.

The module then prompts and reads the following input fields:  
1) source, 2) keywords (to be displayed on the map display),  
3) valid time span, and 4) free-form text describing the environmental conditions.

When the ENTER key is pressed, the module formats the data entered and sends it in an Enter-Manual-Environment-Record message to the ACCESS-ENVIRONMENTAL-DATA process. When the acknowledgement is received, the module sets the display station state to ready and returns control to the main program.

Procedure    MODIFY-MAN-ENV-INFO

Description: This module is called from the main program of the TRANSACT-WATCHSTANDER-REQUESTS process to enable the watchstander to change data stored in a manually-entered environmental data record.

The module first requests the watchstander to specify a sector by entering a sector designation or valid time span. A Get-Manual-Environment-Records message is then sent to the ACCESS-ENVIRONMENTAL-DATA process with the sector designation or the valid time span. The answer contains a list of the qualified records with their corresponding relative record numbers.

If no records were found for the sector, 'NO MANUAL DATA FOR SECTOR' is displayed, the display station state is set to ready, and the module returns control to the main program.

If records were found, a list of the time/date entered and the coordinates is displayed. The watchstander is prompted to select from the displayed list.

When a selection is made, the environmental record data is displayed. The watchstander may change any of the fields displayed by positioning the cursor and typing the correction over the old data. If the NO key is pressed, the record is not changed and the list is again displayed. If the YES key is pressed, a new location may be entered in the same manner as in the ENTER-MAN-ENV-INFO module.

When the ENTER key is pressed, the module sends the modified data and the relative record number in a Modify-Manual-Environment-Record message to the ACCESS-ENVIRONMENTAL-DATA process. When the acknowledgement is received, the module display station state is set to ready and control is returned to the main program.



Procedure DELETE-MAN-ENV-INFO

Description: This module is called from the main program of the TRANSACT-WATCHSTANDER-REQUESTS process to enable the watchstander to delete a manually-entered environmental data record.

The environment record is obtained and selected in the same manner as in the MODIFY-MAN-ENV-INFO module. When the record is selected from the list, its contents are displayed and the watchstander is asked whether he wishes to delete the record.

If the NO key is pressed, the list is again displayed allowing another record to be selected. If the YES key is pressed, the relative record number is sent in a Delete-Manual-Environment-Record message to the ACCESS-ENVIRONMENTAL-DATA process. When the acknowledgement is received, 'RECORD DELETED' is displayed. The module then sets the display station state to ready and returns control to the main program.

#### Procedure ENTER-FORECAST

Description: This module is called from the main program of the TRANSACT-WATCHSTANDER-REQUESTS process to enable the watchstander to enter a weather forecast record into the data base.

A blank form for the forecast data is displayed and the watchstander is prompted to fill in the data fields, which are 1) source, 2) starting time/date, 3) ending time/date, 4) sectors covered, and 5) text. When all data has been entered, the ENTER key is pressed. The module sends the forecast data in an Enter-Forecast-Record message to the ACCESS-ENVIRONMENTAL-DATA process. When the acknowledgement (answer) is received, 'FORECAST ENTERED' is displayed. The module then sets the display station state to ready and returns control to the main program.

#### Procedure MODIFY-FORECAST

Description: This module is called from the main program of the TRANSACT-WATCHSTANDER-REQUESTS process to enable the watchstander to change data in a weather forecast record in the data base.

The weather record is obtained and selected in the same manner as the environment record in the MODIFY-MAN-ENV-INFO module. The selected record is displayed. The watchstander may change fields in the displayed record. When the ENTER key is pressed, the updated record with the relative record number is sent in a Modify-Forecast-Record message to the ACCESS-ENVIRONMENTAL-DATA process. When the answer (acknowledgement) is received, 'FORECAST MODIFIED' is displayed, the display station state is set to ready and control is returned to the main program.



Procedure DELETE-FORECAST

Description: This module is called from the main program of the TRANSACT-WATCHSTANDER-REQUESTS process to enable the watchstander to delete a weather forecast record from the data base.

The weather record is obtained and selected in the same manner as the environment record in the MODIFY-MAN-ENV-INFO module. The selected record is displayed and the watchstander is asked whether he wishes to delete it.

If the NO key is pressed, the list of records is again displayed, allowing another record to be selected. If the YES key is pressed, the relative record number is sent in a Delete-Forecast-Record message to the ACCESS-ENVIRONMENTAL-DATA process. When the answer is received, 'FORECAST DELETED' is displayed, the display station status is set to ready and control is returned to the main program.

#### Procedure ENTER-NOTICE

Description: This module is called from the main program of the TRANSACT-WATCHSTANDER-REQUESTS process to allow the watchstander to enter a notice into the notices file.

The module prompts and inputs the following data items:

1. Notice type
2. Light list number (optional)
3. Sectors covered
4. Navaid designation (optional)
5. Starting date/time
6. Ending date/time
7. Text

When the ENTER key is pressed, the module sends the information entered in an Enter-Notice-Record message to the ACCESS-NOTICES-FILE process. When the acknowledgement to the message is received, 'NOTICE ENTERED' is displayed on the alphanumeric screen, the display station state is set to ready, and the module returns control to the main program.



Procedure    MODIFY-NOTICE

Description: This module is called from the main program of the TRANSACT-WATCHSTANDER-REQUESTS process to enable the watchstander to modify a record in the notices file.

The module requests and inputs the sector and the date/time to specify the notice record. A Get-Notices-Records message containing the selection criteria is sent to the ACCESS-NOTICES-FILE process. The answer contains a list of the qualified notice records. The module then displays a list of the date/time entered for each notice record and prompts the watchstander to select one.

The contents of the selected notice are formatted and displayed. Any of the fields may be changed. When the ENTER key is pressed, a Modify-Notice-Record message is sent to the ACCESS-NOTICES-FILE process. Upon receiving the answer the message 'NOTICE MODIFIED' is displayed and the module returns control to the main program.

If the watchstander does not wish to change the displayed notice, then he may press the NO key. The module will then display the list of qualified notices and prompt him to select a notice.

Procedure DELETE-NOTICE

Description: This module is called from the main program of the TRANSACT-WATCHSTANDER-REQUESTS process to enable the watchstander to delete a record in the notices file.

This module is the same as the MODIFY-NOTICE program except that instead of allowing the watchstander to change fields in the displayed notice record, he is asked whether he is sure that he wishes to delete it. If he responds by pressing the YES key, then a Delete-Notice-Record message is sent to the ACCESS-NOTICES-FILE process. Upon receiving the answer, the message 'NOTICE DELETED' is displayed and the module returns control to the main program.



#### Procedure ENTER-TEMP-ROUTE-SEGMENT

Description: This module is called from the main program of the TRANSACT-WATCHSTANDER-REQUESTS process to enable the watchstander to enter a temporary route segment record into the data base.

The module requests and inputs the attributes of the route segment. These include the segment designation, the minimum depth of MLLW, and the speed limit. In order to associate the temporary route segment with a passage, the watchstander is prompted to identify the vessel in the same manner as in the IDENTIFY-VESSEL module. For each endpoint (of a straight line segment) which the watchstander wishes to define, he is prompted to specify a location in the same manner as in the ENTER-MAN-ENV-INFO module. After each location is specified the module displays the question 'MORE END POINTS?'. If the YES key is pressed, the module prompts for another location to be input. If the NO key is pressed, the data input is sent to the ACCESS-WATERWAY-DATA process in an Enter-Temporary-Route-Segment message. When the answer is received, 'SEGMENT ENTERED' is displayed, the display station state is set to ready, and control is returned to the main program.

## Procedure    MODIFY-TEMP-ROUTE-SEGMENT

Description: This module is called from the main program of the TRANSACT-WATCHSTANDER-REQUESTS process to enable the watchstander to modify a temporary route record.

The module prompts the watchstander to enter the route segment designation. The route segment designation is sent in a Get-Route-Segment-Record message to the ACCESS-WATERWAY-DATA process. The data contained in the answer is displayed. The watchstander may change any of the fields by typing over the old data.

If the YES key is pressed, the list of end point coordinates is displayed. If the NO key is then pressed, the segment data is again displayed. When a point is selected from the list, the watchstander is prompted to select one of the following operations on the end point: 1) replace, 2) insert before (in the list), 3) insert after, or 4) delete. If any of the operations other than deletion is selected, the location of the end point is specified in the same manner as in the ENTER-MAN-ENV-INFO module. The module then displays the updated list of end points. If the delete operation was selected, the point specified does not appear on the display list. When the NO key is pressed, the module again places the initial display on the alphanumeric screen.

When all desired changes have been made, the ENTER key is pressed. The updated temporary record data is then sent in a Modify-Temporary-Route-Segment message to the ACCESS-WATERWAY-DATA process. When the answer is received, 'SEGMENT MODIFIED' is displayed, the display station state is set to ready, and control is returned to the main program.



#### Procedure DELETE-TEMP-ROUTE-SEGMENT

Description: This module is called from the main program of the TRANSACT-WATCHSTANDER-REQUESTS process to enable the watchstander to delete a temporary route segment record.

The module prompts the watchstander to enter the route segment designation. The route segment designation is sent in a Get-Route-Segment-Record message to the ACCESS-WATERWAY-DATA process. The route segment data contained in the answer is displayed. The watchstander is prompted to press the YES key to delete the record, or the NO key to leave the record in the data base.

If the YES key is pressed, the segment designation is sent in a Delete-Temporary-Route-Segment message to the ACCESS-WATERWAY-DATA process. When the answer is received, 'SEGMENT DELETED' is displayed on the alphanumeric screen. The module then sets the display station state to ready and returns control to the main program.

## Procedure ALERT-RESPONSE

Description: This module is called from the main module of the TRANSACT-WATCHSTANDER-REQUESTS process when the watchstander presses the ALERT RESPONSE function key.

The module accesses the WATCHSTANDER-ALERT-QUEUE (WAC) via messages to the MANAGE-ALERT-DISPLAY process. If there are no alerts on the WAC, then a 'NO ALERTS' message is displayed on the alphanumeric screen, the state is set to ready and the procedure returns control to the main program; otherwise, the watchstander is prompted (on the alphanumeric display) to enter the number of one of the alerts displayed on the alert display screen, or else to respond by pressing only the carriage return key. The ALERT-RESPONSE module sends a Display-Alert-Info message with the alert number (or if carriage return was pressed, then the "highest-priority" flag is set) to the MANAGE-ALERT-DISPLAY process. The information concerning the alert will be returned in the answer and displayed on the alphanumeric screen. The ALERT-RESPONSE module then displays a list on the alphanumeric screen of the possible actions, i.e., realert, hold, restore, or cancel. If the watchstander selects one of these alert actions, then a message is sent to the MANAGE-ALERT-DISPLAY process, specifying which function is to be performed on the alert. At any point in the program where input from the watchstander is expected, the SAVE function key or ESCAPE function key may be pressed. If the ESCAPE key is pressed then the module sets the state to ready and returns control to the calling (main) module. If the SAVE function key is pressed, the SAVE-GLOBALS module is called to save the data defining the current state of the display station, the state is set to ready, and control is returned to the main program.



Procedure GET-ENCOUNTERS-INFO

Description: This module is called from the main program of the TRANSACT-WATCHSTANDER-REQUESTS process to allow the watchstander to identify a vessel and to display a list of all other vessels which the indicated vessel is expected to overtake, cross, or meet within a time span indicated by the watchstander.

Procedure GET-ENCOUNTERS-INFO

var valid-list-entry: ('1', '2', '3')

begin

while state <> ready and fct = encounters

do

case state

2: GET-VESSEL-IDENTITY; /\*gets passage record and puts  
internal ID in output msg buffer\*/

3: begin

DISPLAY-ALPHANUMERIC ('VESSEL NOT UNDERWAY');

state: = ready

end;

4 : begin

DISPLAY-ALPHANUMERIC (Choose-Time-Span-Spec);

GET-KEY; /\*get keyboard input\*/

if state = 4 and fct = encounters

then

if key = carriage-return

then /\*read list entry number\*/

if msgbuf.data (1) /\*first data character\*/

and msgbuf.length = 1 /\*only 1 data character\*/

then

case msgbuf.data (1)

'1': state: = 5; /\*specify # minutes\*/

'2': state: = 7; /\*use default time-span\*/

'3': if vessel-routed /\*has an intended route\*/

```

        then
            state: = 6
        else
            DISPLAY-ALPHANUMERIC
                ('VESSEL HAS NO INTENDED ROUTE');
            state = ready;
        end;
    end;
    else DISPLAY-ERROR                ('INVALID RESPONSE')
    end;
    end;
    else DISPLAY- ERROR                ('INVALID RESPONSE'):
    end;
end;
5: begin
    DISPLAY-ALPHANUMERIC (Enter-Look-Ahead-Time-Span);
    GET-KEY; /*get keyboard input*/
    if state = 5 and fct = encounters
    then
        if input time-span in an integer in valid range;
        then
            put time-span in output message buffer;
            Send message to          /*send ENCOUNTERS process
            DETERMINE-ENCOUNTERS;      internal ID and time-span
                                      and wait for answer*/

            state:=7
        else
            DISPLAY-ERROR                ('INVALID RESPONSE');
        end;
    end
end; /*state 5*/
6: begin
    DISPLAY-ALPHANUMERIC (Enter-Waypoint-Designation);
    GET-KEY; /*get keyboard input*/
    if state = 6 and fct = encounters
    then

```



```

if input is valid waypoint designation format;
then
    put waypoint designation in output message
    buffer;

    Send message to DETERMINE-ENCOUNTERS; /*send ENCOUNTERS
                                           process internal ID
                                           and waypoint designa-
                                           tion and wait for
                                           answer*/

    state:=7
    end;
end;
end; /*state 6*/
7: begin
    FORMAT-ENCOUNTER-LIST-DISPLAY; /*using data in answer
                                     buffer from ENCOUNTERS
                                     process, format
                                     display*/

    DISPLAY-ALPHANUMERIC (Encounters-List);
    end; /*state 7*/
end; /*case state*/
end; /*while*/
end; /*procedure GET-ENCOUNTERS-INFO*/

```

\*Procedure GET-VESSEL-IDENTITY

This module is called to obtain the vessel identity from the watchstander. Upon entry, the module displays a numbered list of the ways the vessel may be identified (see Part I, Section 6.2.2). The watchstander types the list number corresponding to the identification method desired. The module then prompts him for the type of identification input required for the specified method. The module sends a Get-Passage-Record message to the ACCESS-PASSAGE-FILE process.

If the watchstander specified the 3-letter wildcard identification, the answer will contain a list of qualified vessel names and disc addresses. The module will display the list of vessel names and prompt the watchstander to select one of the vessels. A Read-Passage-Record message with the disc address of the selected vessel is sent to the ACCESS-PASSAGE-FILE process. The answer contains a copy of the passage record, which this module displays.

If the watchstander specified an identification method other than the 3-letter wildcard, the answer to the Get-Passage-Record message will contain the passage record data, which the GET-VESSEL-IDENTITY module displays on the alphanumeric screen.



Procedure GET-RELATIVE-POSITION-INFO

Description: This module is called from the main program of the TRANSACT-WATCHSTANDER-REQUESTS process to enable the watchstander to obtain the range and relative bearing between two locations which he specifies.

The watchstander is prompted to select one of the identification methods for the origin point: 1) enter latitude and longitude, 2) use the SET function to locate the position on the map display, 3) identify a vessel (as in the GET-VESSEL-IDENTITY module), or 4) enter a navaid designation. When the watchstander chooses from the list of methods, he is then prompted to enter the information necessary for the type of identification selected.

The above procedure is then repeated for the destination location.

When the coordinates for the two positions have been determined, the range and bearing (from origin to destination) are computed using spherical geometry. The range and bearing are displayed on the alphanumeric screen, the display station state is set to ready and the module returns control to the main program.

Procedure GET-CPA-INFO

Description: This module is called from the main program of the TRANSACT-WATCHSTANDER-REQUESTS process to enable the watchstander to obtain the Closest Point of Approach (CPA) between two points, when at least one of the two points is a moving object.

This module obtains watchstander input in the same way as the GET-RELATIVE-POSITION-INFO procedure.

After both positions have been specified and if neither point is moving, a message to that effect is displayed; otherwise, the CPA and time to CPA are calculated. If the time to CPA is greater than the CPA-Time-Threshold (from the system parameters file), a message is displayed indicating that the CPA cannot be computed accurately because of this condition. Otherwise, the CPA position and time are displayed on the alphanumeric screen, and a Display-CPA message is sent to the MANAGE-MAP-DISPLAY process.

Every six seconds the CPA is recomputed, displayed, and sent to the MANAGE-MAP-DISPLAY process. This is repeated until the ESCAPE key is pressed; in which case, a Clear-CPA-Display message is sent to the MANAGE-MAP-DISPLAY process, the display station state is set to ready, and control is returned to the main program.



## Procedure ROUTE-SCHEDULE

Description: This module is called from the main program of the TRANSACT-WATCHSTANDER-REQUESTS process to determine optimal routes and schedules for a vessel.

The vessel is first identified in the same manner as in the MODIFY-PASSAGE module. The watchstander is requested to enter the following information about the vessel : 1) initial waypoint designation, 2) time at initial waypoint, 3) final waypoint designation, 4) speed of vessel, and 5) desired route (a list of route segment designations).

When the ENTER key is pressed, the module determines the possible alternative routes/schedules and orders them according to minimum probable congestion. The best route is displayed as a list of waypoints and the time at each waypoint. If the NO key is pressed, the next best route is displayed.

When the YES key is pressed, the vessel's intended route (in the passage record) will be set to the currently displayed route/schedule. 'VESSEL SCHEDULE/ROUTE ENTERED' is displayed, the display station state is set to ready, and control is returned to the main program.

The ROUTE-SCHEDULE procedure utilizes numerous system procedures to determine the optimum routes and schedules which minimize congestion and potential critical encounters. At a minimum the algorithms used by this module will include: GET-LOCAL-TRAFFIC (Part I, Section 14.2.1.2); DETECT-EXCESSIVE-CONGESTION (Part II, Section 3.2.6); PREDICT-ENCOUNTERS (Part II, Section 3.2.7); GET-RELATIVE-POSITION-INFO (Part II, pg. 4-63 and Appendix A.1); and GET-CPA-INFO (Part II, pg. 4-64 and Appendix A.2). These algorithms will be used to satisfy site specific requirements based on the results of detailed vessel traffic management studies.

Procedure    ACKNOWLEDGE

Description: This module is called from the main program of the TRANSACT-WATCHSTANDER-REQUESTS process to enable the watchstander to respond to the ACTION-REQUIRED indicator.

The watchstander is prompted to enter the identification number of the message on the action-required list (as displayed on the action-required display screen). To select the message on the top of the list, the carriage RETURN key is pressed. The module sends the message number in a Get-A-R-L-Entry to the MANAGE-ACTION-REQUIRED-LIST process.

The answer contains the information from the Action-Required-List entry corresponding to the specified message number. This information is displayed on the alphanumeric screen. The display station state is set to ready and control is returned to the main program.



#### Procedure RECORD-SCENARIO

Description: This module is called from the main program of the TRANSACT-WATCHSTANDER-REQUESTS process to record data for a simulation.

The data required to create a scenario record is prompted for and input. It is then sent as a Record-Scenario message to the SET-UP-SCENARIO process. When an answer is received the display station state is set to ready and control is returned to the main program.

#### Procedure MODIFY-SCENARIO

Description: This module is called from the main program of the TRANSACT-WATCHSTANDER-REQUESTS process to enable the watchsupervisor to modify data in a Scenario-Record stored in the data base.

The scenario record name is entered. The scenario record is read, and its contents displayed. Fields in the record may be altered by typing over existing data. When the ENTER key is pressed, the modified record is sent in a Modify-Scenario message to the SET-UP-SCENARIO process. When an answer is received the display station state is set to ready and control is returned to the main program.

#### Procedure DELETE-SCENARIO

Description: This module is called from the main program of the TRANSACT-WATCHSTANDER-REQUESTS process to enable the watch supervisor to delete a scenario record from the data base.

The scenario record name is entered and sent to the SET-UP-SCENARIO process in a Delete-Scenario message. When an answer is received the display station status is set to ready and control is returned to the main program.

#### Procedure INIT-SCENARIO

Description: This module is called from the main program of the TRANSACT-WATCHSTANDER-REQUESTS process to enable the watch supervisor to initialize data before starting the playback of a scenario.

The following information is prompted for, input, and sent in an Initialize-Scenario message to the PLAYBACK-SCENARIO process:  
1) scenario record name, 2) ID of display station to receive simulated data and 3) a flag indicating whether to pre-load artificial vessel and passage files.

When an answer is received then the display station state is set to ready and control is returned to the main program.

#### Procedure START-SCENARIO

Description: This module is called from the main program of the TRANSACT-WATCHSTANDER-REQUESTS process to enable the watch supervisor to start the playback of a scenario.

The following information is prompted for, input, and sent in a Start-Scenario message to the PLAYBACK-SCENARIO process: 1) scenario name, 2) relative time at which to begin scenario, and 3) relative playback speed (16 or 60 times the recording speed).

When an answer is received the display station state is set to ready and control is returned to the main program.



#### 4.2 MANAGE-MAP-DISPLAY Process

**Brief Description:** The primary function of this process is to keep the map display updated. In addition, it can be used with a cursor positioning device to return the identity of a displayed vessel, or a set of coordinates specified by where the cursor is positioned. The rate of update of the display should not be slower than once every 6 seconds, but this is dependent on the rate at which the new information is received from other processes

MANAGE-MAP-DISPLAY control/data paths are shown in Figure 4-2A/B/C.

**Relationship to Functional Description:**  
Appendix 8 - 5.5.1 Verification

##### **Message Types:**

This process is described in terms of the different types of messages which can be received by the process. For each message type, the function to be performed, the input information in the message, and the output information (if required) in the answer to the message are described below.

##### **MESSAGE TYPE: Get-Location**

**FUNCTION:** Return cursor coordinates when SET function key is pressed. Place bold X on map display at position.

**INPUT:** None

**ANSWER:** Internal coordinates of position

##### **MESSAGE TYPE: Get-Identification**

**FUNCTION:** Return vessel or navaid internal ID of object nearest to cursor position when HOOK function key is pressed. Hooked object is circled on map display.

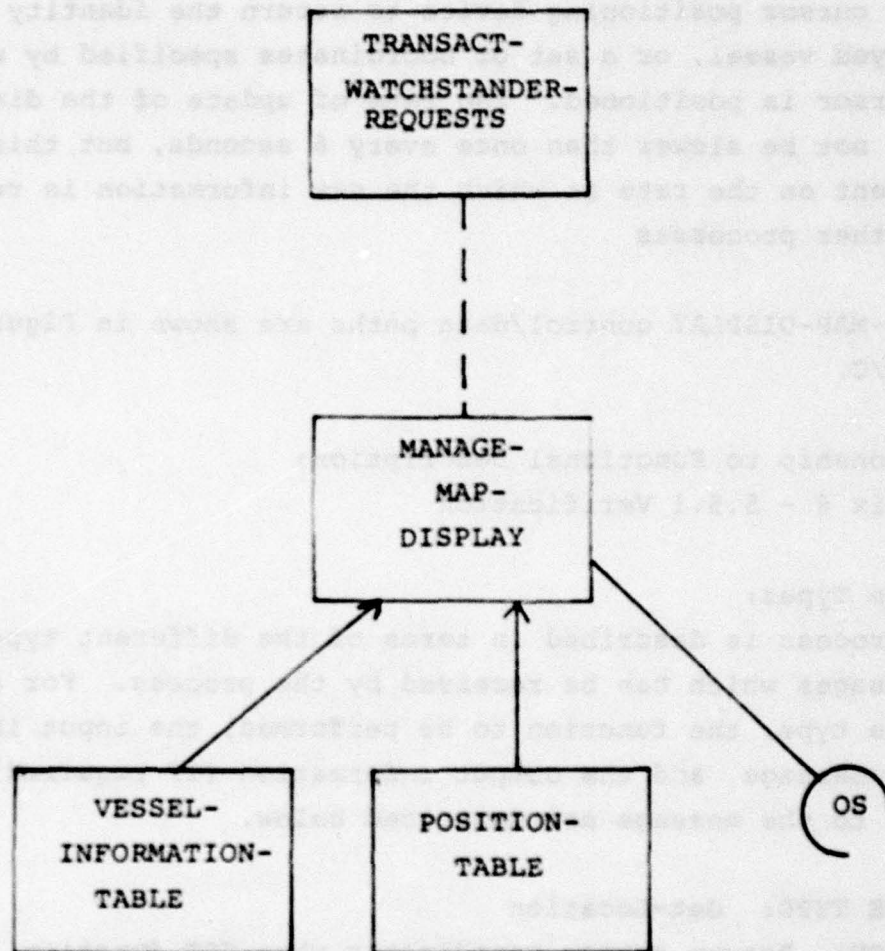


Figure 4-2A. MANAGE-MAP-DISPLAY



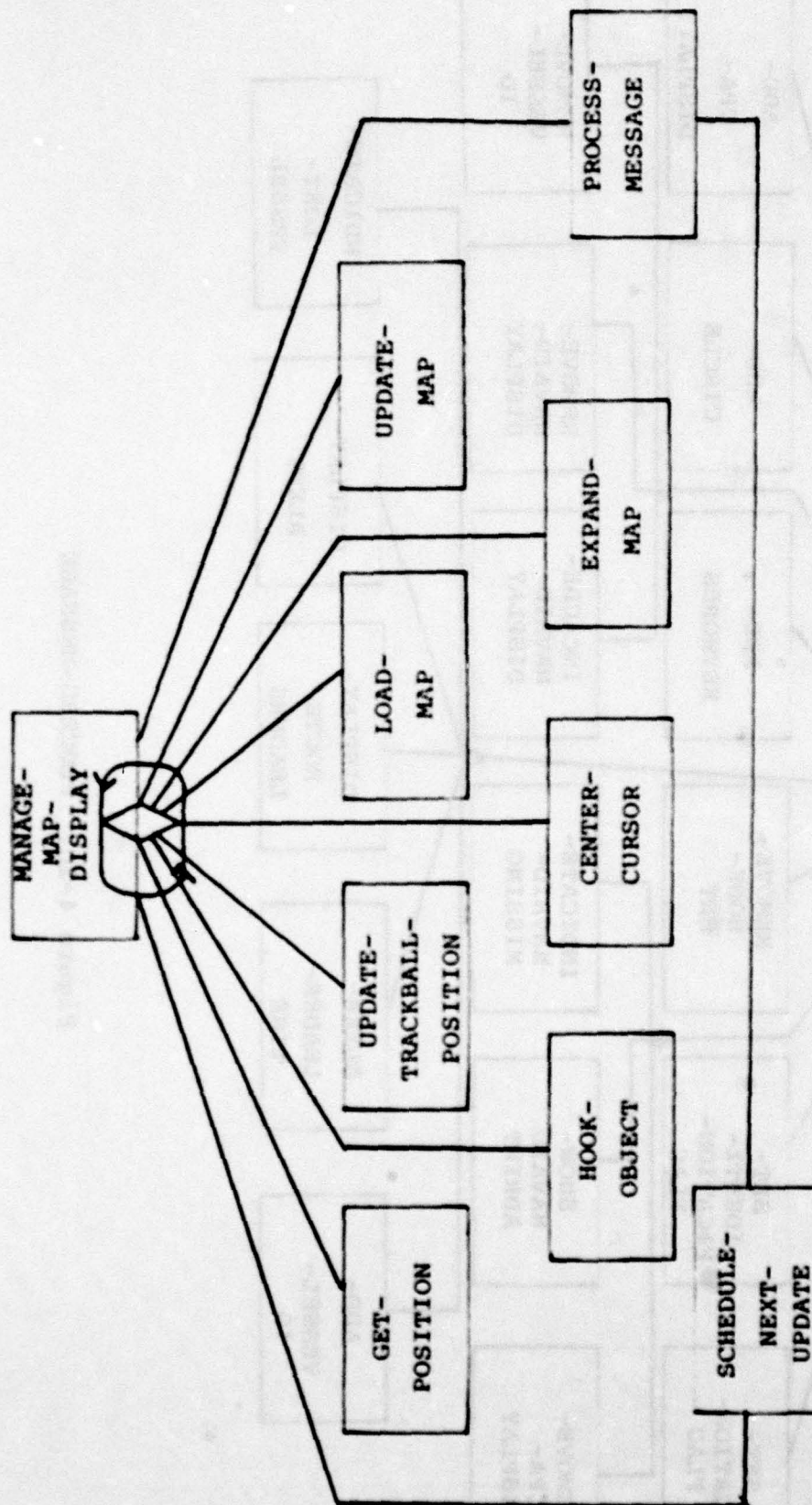


Figure 4-2B. MANAGE-MAP-DISPLAY

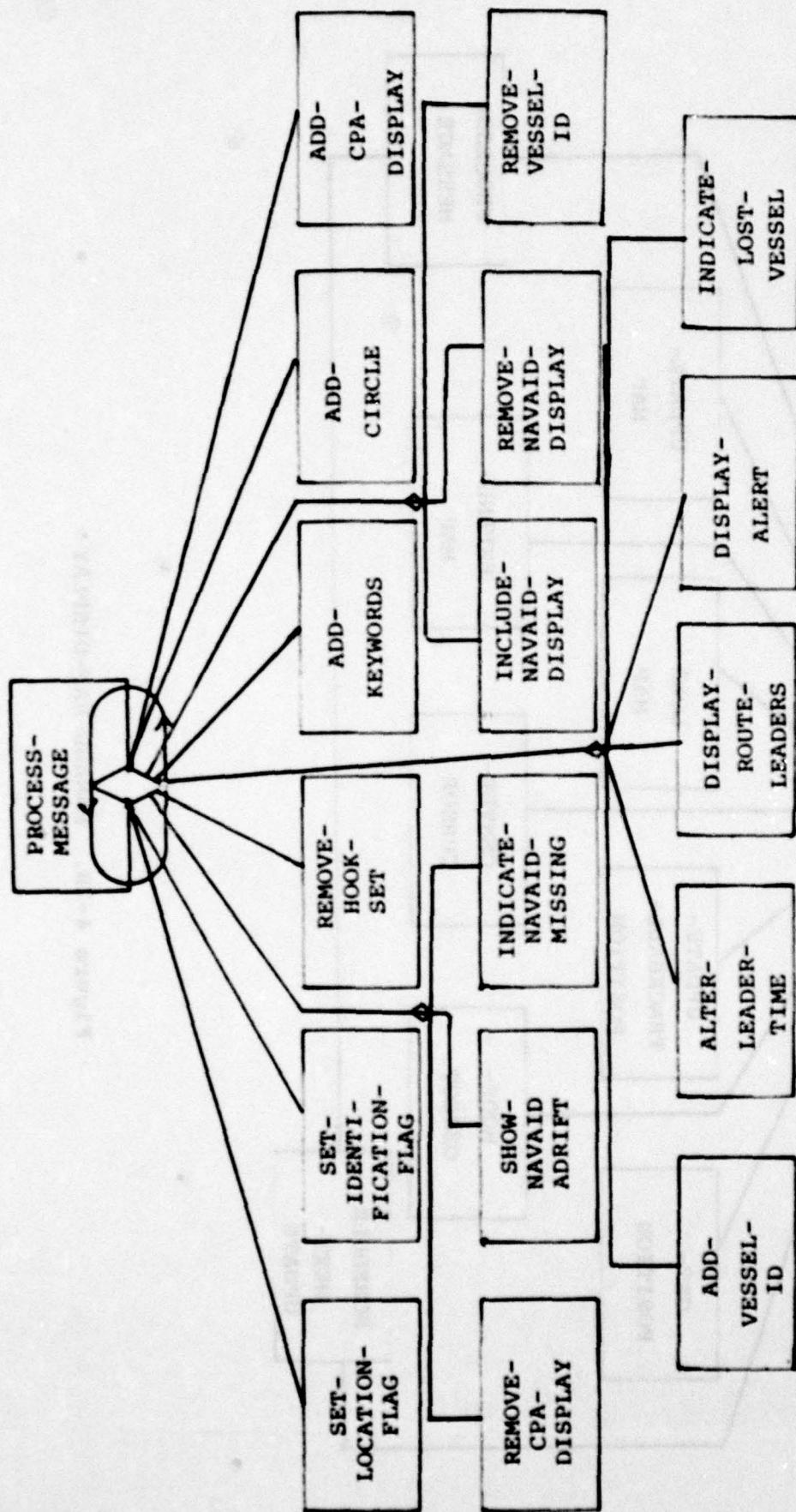


Figure 4-2C. PROCESS-MESSAGE



INPUT: None

ANSWER: 1) Internal ID  
2) Vessel or Navaid ID Code (if identified)

MESSAGE TYPE: Clear-Hook-And-Set

FUNCTION: Remove X on map display from previous SET.  
Remove circle(s) on map display from previous  
Hook or alert.

INPUT: None

ANSWER: Acknowledgment

MESSAGE TYPE: Display-Keywords

FUNCTION: The specified keywords as placed on the map  
display at the given position.

INPUT: 1) List of keywords  
2) Internal Coordinates of position

ANSWER: Acknowledgment

MESSAGE TYPE: Circle-Object

FUNCTION: The specified vessel or navaid is circled  
until a Clear-Hook-and-Set message is received.

INPUT: Internal ID of object

ANSWER: Acknowledgment

MESSAGE TYPE: Mark-Navaid-Adrift

FUNCTION: Both the proper position and the present position  
of the navaid are displayed. A leader line  
indicating course and speed is displayed at  
the present position.

INPUT: Internal ID of navaid

ANSWER: Acknowledgment

MESSAGE TYPE: Mark-Navaid-Missing

FUNCTION: A special symbol is displayed at the original  
navaid position.

INPUT: Internal ID of navaid

ANSWER: Acknowledgment

MESSAGE TYPE: Display-CPA

FUNCTION: Display the changing relative CPA positions.

INPUT: For both positions

- a) Flag indicating whether moving object
- b) If moving object, internal ID;  
otherwise, coordinates
- c) If moving object, position at CPA.

ANSWER: Acknowledgement

MESSAGE: Clear-CPA-Display

FUNCTION: Remove the CPA display.

INPUT: None

ANSWER: Acknowledgement.

MESSAGE TYPE: Add-Navaid-Display

FUNCTION: The specified navaid is included in the map  
display.

INPUT: 1) Internal ID

2) Watch Circle Radius

ANSWER: Acknowledgment

MESSAGE TYPE: Suppress-Navaid-Display

FUNCTION: The specified navaid is deleted from the map  
display.

INPUT: Internal ID

ANSWER: Acknowledgment



MESSAGE TYPE: Change-Leader-Time

FUNCTION: The leader line for each moving vessel is changed to indicate the position at the specified number of minutes (from 1 to 15) in the future.

INPUT: Number of minutes for leader line time.

ANSWER: Acknowledgment

MESSAGE TYPE: Display-Intended-Routes

FUNCTION: A 6 second temporary display will show the intended route of all tracked vessels instead of the dead reckoned leader line.

INPUT: None

ANSWER: Acknowledgment

MESSAGE TYPE: Display-Alert-Situation

FUNCTION: The vessels/navaids affected by the alert will be displayed in a manner to indicate their involvement (e.g., circle affected vessels). Fixed points (e.g., point of predicted grounding) will be displayed by a special symbol. The display differs according to the type of alert.

INPUT: 1) Type of alert  
2) Internal coordinates of location(s)  
3) Internal IDs of affected vessels and navaids  
4) Other alert information (varies with type of alert).

ANSWER: Acknowledgment

MESSAGE TYPE: Display-Vessel-Lost

FUNCTION: Vessel symbol blinks on map display at last known position.

INPUT: Internal ID

ANSWER: Acknowledgment

Each of the above functions would be executed by a separate procedure within the MANAGE-MAP-DISPLAY process.

#### Data Structures:

The MANAGE-MAP-DISPLAY process has direct read-only access to the Vessel-Information-Table, which is maintained by the MANAGE-VESSEL-INFORMATION-TABLE process, and to the Position-Table which is maintained by the MANAGE-POSITION-TABLE process. The Position-Table contains the location, course, speed, and size for all vessels and nav aids, both tracked and untracked. The Vessel-Information-Table contains for each vessel the internal ID, the vessel ID code (external map label), the status, the vessel draft, the intended route, the current route segment, tracked/untracked flag and hazard processing exemption flags.

The Map-Structure-Table contains all the permanent map information, including map outline, landmarks, traffic lanes, proper nav aid positions, and precautionary areas. It is loaded from disc when a different map is requested (i.e., when the map selection key is pressed).

The Display-Command-Table contains all the vector and symbol commands in a format required by the graphics (map) display. It is created and maintained by processing the combined data of the Map-Structure-Table, the Vessel-Information-Table, and the Position-Table.



Process MANAGE-MAP-DISPLAY

begin

SCHEDULE-NEXT-UPDATE;

Request the operating system to set the following event flags  
on input from the corresponding device: Map-Key-Event-Flag,  
Expand-Key-Event-Flag, Set-Key-Event-Flag, Hook-Key-Event-Flag,  
Trackball-Event-Flag, Center-Cursor-Event-Flag;

Request the operating system to set the Message-Event-Flag  
whenever a message is received;

while true do

begin

if no event flags are set

then wait for any event flag to be set;

case event flag of /\*functions listed in order of priority\*/

Set-Key-Event-Flag: GET-POSITION;

Hook-Key-Event-Flag: HOOK-OBJECT;

Trackball-Event-Flag: UPDATE-TRACKBALL-POSITION;

Center-Cursor-Event-Flag: CENTER-CURSOR;

Map-Key-Event-Flag: LOAD-MAP;

Expand-Key-Event-Flag: EXPAND-MAP;

Update-Map-Event-Flag: UPDATE-MAP;

Message-Event-Flag: PROCESS-MESSAGE;

end;

end;

end

Procedure SCHEDULE-NEXT-UPDATE

Description: Send a message to the ACCESS-SYSTEM-PARAMETERS process requesting the current Map-Update-Cycle-Time. Wait for the answer. Request the operating system to set the Update-Map-Event-Flag at Map-Update-Cycle-Time from now.



#### Procedure LOAD-MAP

Description: Get number of MAP key. Send a message to the ACCESS-MAP-FILE process to read the corresponding map from the data base. The answer contains the map data which is placed in the Map-Structure-Table. The module then converts this data into graphics vectors and symbols which are placed in the Display-Command-Table. Call UPDATE-MAP module to place vessel and navaid data in the Display-Command-Table. Clear the Map-Key-Event-Flag.

#### Procedure EXPAND-MAP

Description: Get magnification factor. Get last cursor position. Compute coordinate limits for expanded section. Extract data from Map-Structure-Table which lies within coordinate limits. Convert extracted data into graphics vectors and symbols, which are placed in the Display-Command-Table. Call the UPDATE-MAP module to place vessel and navaid data in the Display-Command-Table. Clear the Expand-Key-Event-Flag.

#### Procedure UPDATE-MAP

Description: Clear the Update-Map-Event-Flag. Call the SCHEDULE-NEXT-UPDATE module. Get the leader line time. For each entry in the Position-Table which lies within the current map coordinate limits, get the entry (if any) from the Vessel-Information-Table with the same internal ID. Use the data from the entries to create the vectors and symbols necessary to display the vessel or navaid with its symbol, label and leader line (as required). Merge these graphics commands into the Display-Command-Table.

#### Procedure GET-POSITION

Description: Get the last cursor screen position. Convert this position to internal system coordinates based on the current map display coordinate limits. Place a bold "X" symbol in the Display-Command-Table at the cursor screen position. If the Return-Location-Flag is set then send the system coordinates in an answer to the process indicated by the Location-Answer-Process-Name. Clear the Return-Location-Flag. Clear the Set-Key-Event-Flag.

#### Procedure HOOK-OBJECT

Description: Get the last cursor screen position. Convert the position to internal system coordinates based on the current map display coordinate limits. Search the Position-Table for the vessel or navaid closest to the hooked position. Place a circle around the object screen symbol in the Display-Command-Table. If the Return-ID-Flag is set then send the internal ID and external ID to the process indicated by the ID-Answer-Process-Name. Clear the Return-ID-Flag. Clear the Hook-Key-Event-Flag.

#### Procedure UPDATE-TRACKBALL-POSITION

Description: If change in X coordinate then add change to X coordinate of cursor position. If change in Y coordinate then add change to Y coordinate of cursor position. Change cursor position vector in Display-Command-Table. Clear the Trackball-Event-Flag.

#### Procedure CENTER-CURSOR

Description: Set the cursor position to the center of the map display screen. Set the cursor position vector in the Display-Command-Table. Clear the Center-Cursor-Event-Flag.



Procedure PROCESS-MESSAGE

begin

case message type of

Get-Location: SET-LOCATION-FLAG  
Get-Identification: SET-IDENTIFICATION-FLAG  
Clear-Hook-And-Set: REMOVE-HOOK-SET  
Display-Keywords: ADD-KEYWORDS  
Circle-Object: ADD-CIRCLE  
Display-CPA: ADD-CPA-DISPLAY  
Clear-CPA-Display: REMOVE-CPA-DISPLAY  
Mark-Navaid-Adrift: SHOW-NAVAID-DRIFT  
Mark-Navaid-Missing: INDICATE-NAVAID-MISSING  
Add-Navaid-Display: INCLUDE-NAVAID-DISPLAY  
Suppress-Navaid-Display: REMOVE-NAVAID-DISPLAY  
Suppress-Vessel-Label: REMOVE-VESSEL-ID  
Restore-Vessel-Label: ADD-VESSEL-ID  
Change-Leader-Time: ALTER-LEADER-TIME  
Display-Intended-Routes: DISPLAY-ROUTE-LEADERS  
Display-Alert-Situation: DISPLAY-ALERT  
Display-Vessel-Lost: INDICATE-LOST-VESSEL

end;

Send an answer acknowledging the message, clear the Message-  
Event-Flag;

end

Procedure SET-LOCATION-FLAG

Description: Set the Return-Location-Flag. Move the name of the process from which the message was received to the Location-Answer-Process-Name so that the GET-POSITION module will know where to send the location when the SET key is pressed.

Procedure SET-IDENTIFICATION-FLAG

Description: Set the Return-ID-Flag. Move the name of the process from which the message was received to the ID-ANSWER-Process-Name so that the HOOK-OBJECT module will know where to send the vessel/navaid IDs when the HOOK key is pressed.

Procedure REMOVE-HOOK-SET

Description: Remove the "hook" circle (if any) and the "set" bold "X" from the Display-Command-Table.

Procedure ADD-KEYWORDS

Description: Add the keywords contained in the message to the Display-Command-Table at the position indicated in the message.

Procedure ADD-CPA-DISPLAY

Description: Make the following additions to the Display-Command-Table: 1) circle each of the two objects specified in the message 2) for each moving object alter the leader line to run to the position at CPA, and 3) place a dashed line between the 2 positions at CPA.

Procedure REMOVE-CPA-DISPLAY

Description: Remove from the Display-Command-Table the vectors inserted by the ADD-CPA-DISPLAY procedure.



Procedure SHOW-NAVAID-DRIFT

Description: Add a leader line to the navaid symbol at the present navaid position. This leader indicates the direction and speed of the drift.

Procedure INDICATE-NAVAID-MISSING

Description: At the proper navaid position in the Display-Command-Table, replace the normal symbol with a special symbol.

Procedure INCLUDE-NAVAID-DISPLAY

Description: Add the navaid specified in the message to the display by placing in the Display-Command-Table the navaid designation (alphanumeric label) and the symbol indicating its position.

Procedure REMOVE-NAVAID-DISPLAY

Description: Delete from the Display-Command-Table the label and position symbol of the specified navaid.

Procedure REMOVE-VESSEL-ID

Description: Delete from the Display-Command-Table the label of the vessel specified in the message.

Procedure ADD-VESSEL-ID

Description: Restore to the Display-Command-Table the label of the vessel specified in the message.

Procedure ALTER-LEADER-TIME

Description: The Leader-Time (used in the calculation of the leader line length for all moving vessels) is changed to the number of minutes specified in the message.

Procedure DISPLAY-ROUTE-LEADERS

Description: Temporarily alter the Display-Command-Table to display the leader lines along intended routes only. After the next map update cycle, the Display-Command-Table is switched back to display the normal dead-reckoned leader lines.

Procedure DISPLAY-ALERT

Description: The Display-Command-Table will be altered as follows. The vessels and nav aids specified will be circled. The locations specified will be displayed by a special symbol.

Procedure INDICATE-LOST-VESSEL

Description: In the Display-Command-Table the vessel symbol of the specified vessel will be caused to blink.

Procedure ADD-CIRCLE

Description: Get the internal ID from the message. Place a circle around the object (with the specified internal ID) screen symbol in the Display-Command-Table.



#### 4.3 MANAGE-ALERT-DISPLAY Process

Brief Description: A copy of this process exists in each display station processor. It manages a list of alerts queued to the watchstander and displays the alerts as described in Part I, Section 13.7. It is also responsible for turning on and off the alert alarm.

MANAGE-ALERT-DISPLAY control/data paths are shown in Figure 4-3A/B.

Relationship to Function Description:  
Appendix 8 - 5.6.2

##### Message Types:

The MANAGE-ALERT-DISPLAY process is responsible for maintaining the WATCHSTANDER-ALERT-QUEUE, and for displaying information of each alert entry of the W-A-Q on the alert display screen. These functions are performed in response to messages received from the POST-ALERTS process (in the main processor) and the TRANSACT-WATCHSTANDER-REQUESTS (watchstander alert responses). These messages are described below.

MESSAGE TYPE: Display-Alert-Info (Alert Response)  
SENDING PROCESS: TRANSACT-WATCHSTANDER-REQUESTS  
FUNCTION: Return alert information to be displayed on alpha-numeric screen. Set 'time of last watchstander response to alert' to current time. Turn off alert alarm.  
INPUT: Alert Number or Highest Priority Flag  
ANSWER: Copy of W-A-Q alert entry

MESSAGE TYPE: Realert  
SENDING PROCESS: TRANSACT-WATCHSTANDER-REQUESTS  
FUNCTION: Set alert entry status to 'Realert', and set time of day to realert watchstander, i.e., to turn on alert alarm.

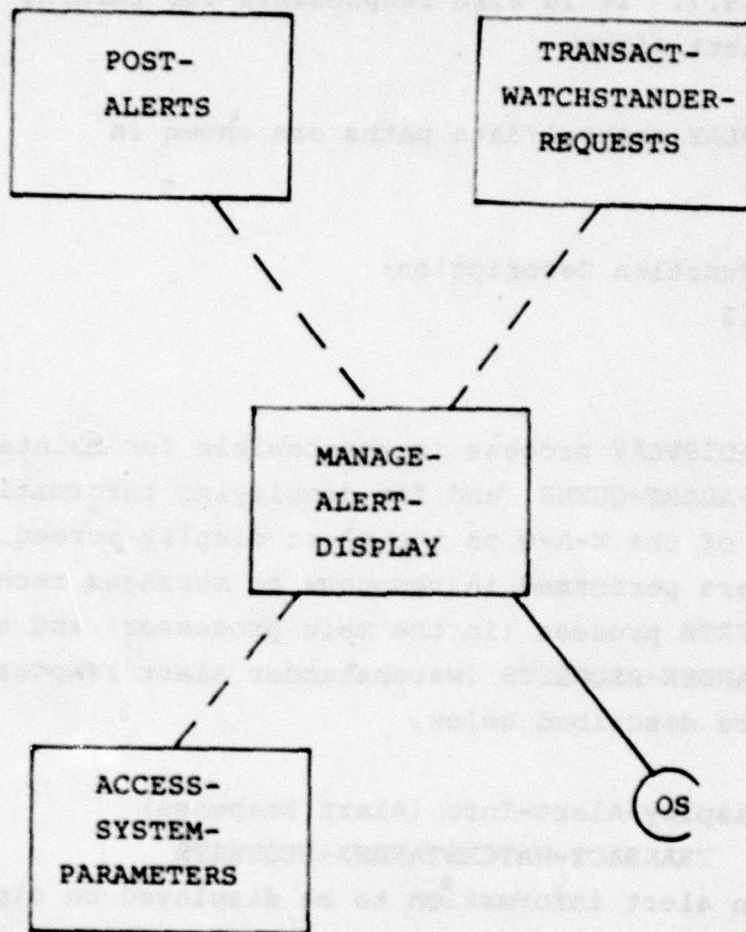


Figure 4-3A. MANAGE-ALERT-DISPLAY



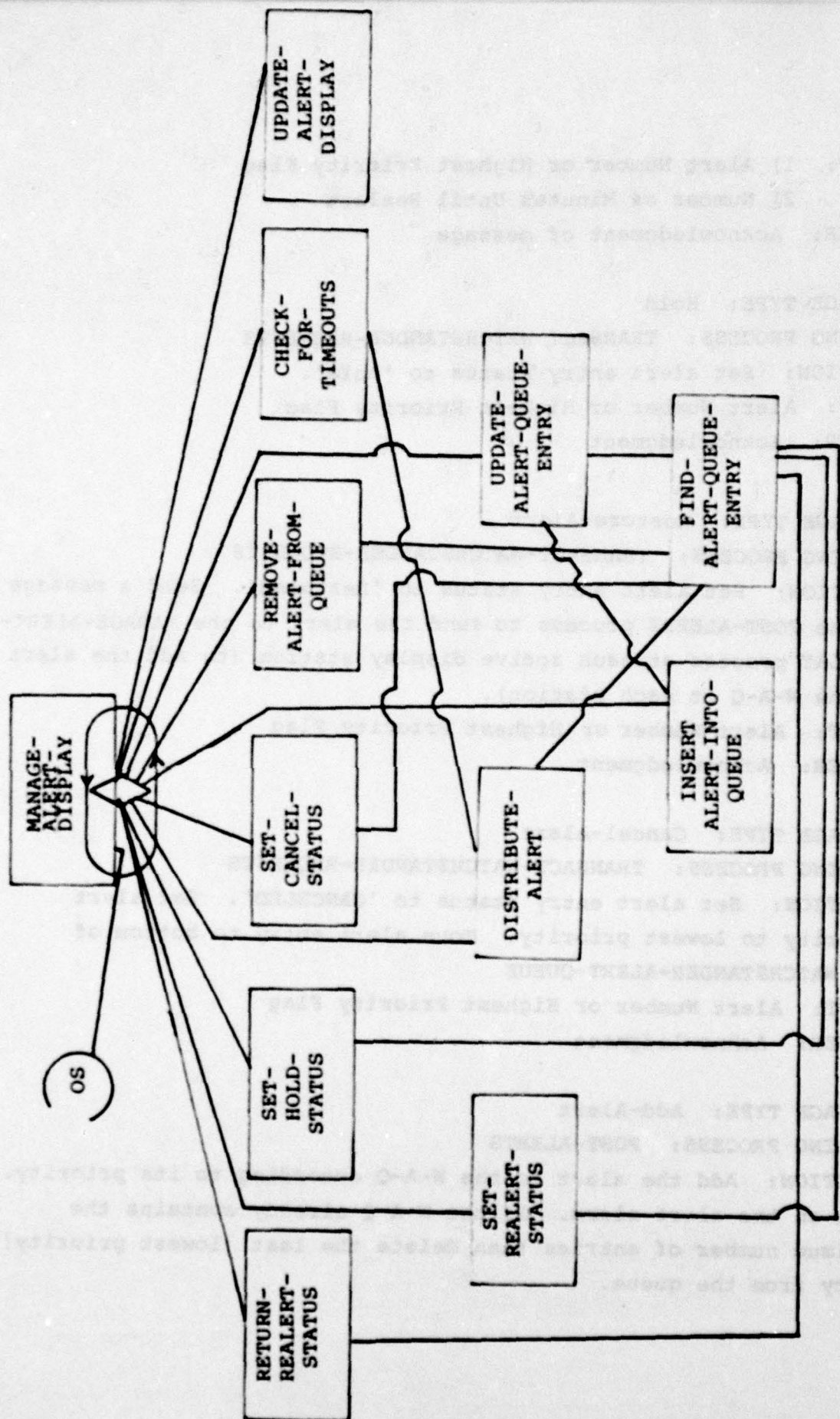


Figure. 4-3B. MANAGE-ALERT-DISPLAY

INPUT: 1) Alert Number or Highest Priority Flag  
2) Number of Minutes Until Realert

ANSWER: Acknowledgment of message

MESSAGE TYPE: Hold

SENDING PROCESS: TRANSACT WATCHSTANDER-REQUESTS

FUNCTION: Set alert entry status to 'hold'.

INPUT: Alert Number or Highest Priority Flag

ANSWER: Acknowledgment

MESSAGE TYPE: Restore-Alert

SENDING PROCESS: TRANSACT-WATCHSTANDER-REQUESTS

FUNCTION: Set alert entry status to 'Restored'. Send a message to the POST-ALERTS process to send the alert to the MANAGE-ALERT-DISPLAY process at each active display station (to add the alert to the W-A-Q at each station).

INPUT: Alert Number or Highest Priority Flag

ANSWER: Acknowledgment

MESSAGE TYPE: Cancel-Alert

SENDING PROCESS: TRANSACT-WATCHSTANDER-REQUESTS

FUNCTION: Set alert entry status to 'CANCELED'. Set alert priority to lowest priority. Move alert entry to bottom of the WATCHSTANDER-ALERT-QUEUE

INPUT: Alert Number or Highest Priority Flag

ANSWER: Acknowledgment

MESSAGE TYPE: Add-Alert

SENDING PROCESS: POST-ALERTS

FUNCTION: Add the alert to the W-A-Q according to its priority. Turn on the alert alarm. If the W-A-Q already contains the maximum number of entries then delete the last (lowest priority) entry from the queue.



INPUT: 1) Alert Number  
2) Alert Type  
3) Latest Time for Effective Action  
4) Alert Status (NEW or RESTORED)  
5) Waterway Sector  
6) Alert-Specific Information (dependent upon alert type)

ANSWER: Acknowledgment

MESSAGE TYPE: Delete-Alert

SENDING PROCESS: POST-ALERTS

FUNCTION: Delete the alert from the W-A-Q

INPUT: Alert Number

ANSWER: Acknowledgment

MESSAGE TYPE: Update-Alert

SENDING PROCESS: POST-ALERTS

FUNCTION: Update the dynamic information in an alert entry already in the W-A-Q. If priority is changed then position in W-A-Q may need to be changed. If the entry is not already in the W-A-Q, then add it to the queue as in the Add-Alert function.

INPUT: 1) Alert Number  
2) Alert Type  
3) Time Remaining for Effective Action  
4) Alert Status  
5) Waterway Sector  
6) Alert-Specific Information

ANSWER: Acknowledgment

For each type of message received, there is a procedure (subroutine) which executes the requested function. In addition to actions initiated by the receiving of messages from other processes, the MANAGE-ALERT-DISPLAY process keeps track of the time since the

last watchstander response to an alert and the priority. When both of these have exceeded their thresholds, a message is sent to the POST-ALERTS process to add the alert to the MANAGE-ALERT-DISPLAY process at each display station (to add the alert to every W-A-Q).

#### Data Structures:

The major data structure is the Watchstander-Alert-Queue (W-A-Q). This queue is a singly-linked list of alert entries in order of priority (risk factor). Each entry contains the following information:

- 1) Alert Number (Identifies alert)
- 2) Priority (risk factor)
- 3) Type of alert
- 4) Alert Status (NEW, REALERT, HOLD, RESTORED, or CANCELLED)
- 5) Latest Time for Effective Action
- 6) Time of Last Watchstander Response to Alert
- 7) Time at Which to Realert Watchstander
- 8) Waterway Sector
- 9) Alert-Specific Information (to be displayed on alphanumeric display screen upon watchstander response to this alert).

The dynamic information in the W-A-Q entries is kept updated with information in messages from the POST-ALERTS and TRANSACT-WATCHSTANDER-REQUESTS processes. This dynamic information includes the priority, status, time remaining for effective actions, time since last watchstander response, and alert-specific information.



Process    MANAGE-ALERT-DISPLAY

begin

while true do

begin

if no message is queued

then wait for a message;

case message type of

        Display-Alert-Info: RETURN-ALERT-INFO;

        Realert: SET-REALERT-STATUS;

        Hold: SET-HOLD-STATUS;

        Restore-Alert: DISTRIBUTE-ALERT;

        Cancel-Alert: SET-CANCEL-STATUS

        Add-Alert: INSERT-ALERT-INTO-QUEUE;

        Delete-Alert: REMOVE-ALERT-FROM-QUEUE;

        Update-Alert: UPDATE-ALERT-QUEUE-ENTRY

end

      Send answer to message;

      CHECK-FOR-TIMEOUTS;

      UPDATE-ALERT-DISPLAY;

end

end

Procedure CHECK-FOR-TIMEOUTS

begin

Send a message to the ACCESS-SYSTEM-PARAMETERS process  
requesting the Alert-Timeout-Priority and the Alert-  
Response-Timeout;

Wait for the answer;

repeat until the Alert-Timeout-Priority is higher than  
the priority of the W-A-Q entry

begin

Get the next entry in the W-A-Q;

Get the current time from the operating system;

if the time of last watchstander response to alert  
plus the Alert-Response-Timeout is earlier than the  
current time of day

then DISTRIBUTE-ALERT;

end

end

Procedure UPDATE-ALERT-DISPLAY

begin

for each entry in the W-A-Q do

Move the alert number, alert type, latest time of day  
for effective action, sector, and status from the W-A-Q  
entry into the next line of the alert display buffer;

Request the operating system to write the alert display  
buffer to the alert display screen;

end



Procedure UPDATE-ALERT-QUEUE-ENTRY

begin

FIND-ALERT-QUEUE-ENTRY;

if found

then

begin

Move the alert information fields from the message into  
the W-A-Q entry;

if the alert priority is changed

then move the entry to its proper position in the W-A-Q  
(entries are ordered by priority);

end

else

INSERT-ALERT-INTO-QUEUE

end

Procedure REMOVE-ALERT-FROM-QUEUE

begin

FIND-ALERT-QUEUE-ENTRY;

Delete the entry from the W-A-Q;

end

Procedure INSERT-ALERT-INTO-QUEUE

begin

if the W-A-Q contains the maximum number of entries  
then delete the lowest priority entry from the W-A-Q;

Create an entry in the W-A-Q positioned according to  
the alert priority;

Move the contents of the message to the new alert queue  
entry;

Turn on the alert alarm;

end

Procedure SET-CANCEL-STATUS

begin

FIND-ALERT-QUEUE-ENTRY;

In the W-A-Q entry, set the status to 'cancelled' and the  
priority to the lowest priority;

Move the alert entry to the bottom of the W-A-Q;

end

Procedure DISTRIBUTE-ALERT

begin

FIND-ALERT-QUEUE-ENTRY;

In the W-A-Q entry set the status to 'restored';

Send a message to the POST-ALERTS process requesting it  
to send a copy of the alert to the MANAGE-ALERT-DISPLAY  
process at each active display station;

Wait for the answer;

end



Procedure SET-HOLD-STATUS

begin

FIND-ALERT-QUEUE-ENTRY;

In the W-A-Q entry set the status to 'hold';

end

Procedure SET-REALERT-STATUS

begin

FIND-ALERT-QUEUE-ENTRY;

In the W-A-Q entry set the status to 'Realert';

Request the current time of day from the operating system;

Add to the time of day the 'number of minutes until realert'  
and place the sum in the alert queue entry field, 'time at  
which to realert watchstander';

end

Procedure RETURN-ALERT-INFO

begin

FIND-ALERT-QUEUE-ENTRY;

Place in the answer buffer the alert information to be  
displayed on the alphanumeric screen;

Request the current time of day from the operating system  
and place it in the W-A-Q entry field for 'time of last  
watchstander response to alert';

Turn off the alert alarm;

end

\*Procedure FIND-ALERT-QUEUE-ENTRY

begin

Look at the alert number in the message buffer;

if the alert number = 0

then

Set the W-A-Q entry pointer to the address of the first entry on the W-A-Q

else

Search the W-A-Q until the alert number in the W-A-Q entry = the alert number in the message buffer, then set the W-A-Q entry pointer to the address of the W-A-Q entry;

end



#### 4.4 MANAGE-ACTION-REQUIRED-LIST Process

The MANAGE-ACTION-REQUIRED-LIST (M-A-R-L) process is responsible for maintaining the ACTION-REQUIRED-LIST (A-R-L), a queue of items requiring action by the watchstander. It also controls the ACTION REQUIRED indicator light. There is a copy of the M-A-R-L and the A-R-L in each display processor.

MANAGE-ACTION-REQUIRED-LIST control/data paths are shown in Figure 4-4A/B.

Relationship to Functional Description:

Appendix - 5.5.1 Verification

Message Types:

This process performs operations on the A-R-L and the ACTION-REQUIRED indicator light, in response to messages received from other processes. The operations for each type of message received are described below.

MESSAGE TYPE: Add-Entry-to-List

SENDING PROCESS: Any process

FUNCTION: Assign an identification number to the message, and add the message to the bottom of the A-R-L. Blink the ACTION-REQUIRED indicator for 5 seconds, then leave it turned on (steadily).

INPUT: A one-line message summarizing the problem is required. Additional information may be supplied, according to the type of problem.

ANSWER: Acknowledge the message.

MESSAGE TYPE: Get-ARL-Entry

SENDING PROCESS: TRANSACT-WATCHSTANDER-REQUESTS

FUNCTION: Return contents of specified A-R-L entry. Remove entry from A-R-L. If A-R-L is now empty, turn off Action-Required indicator.

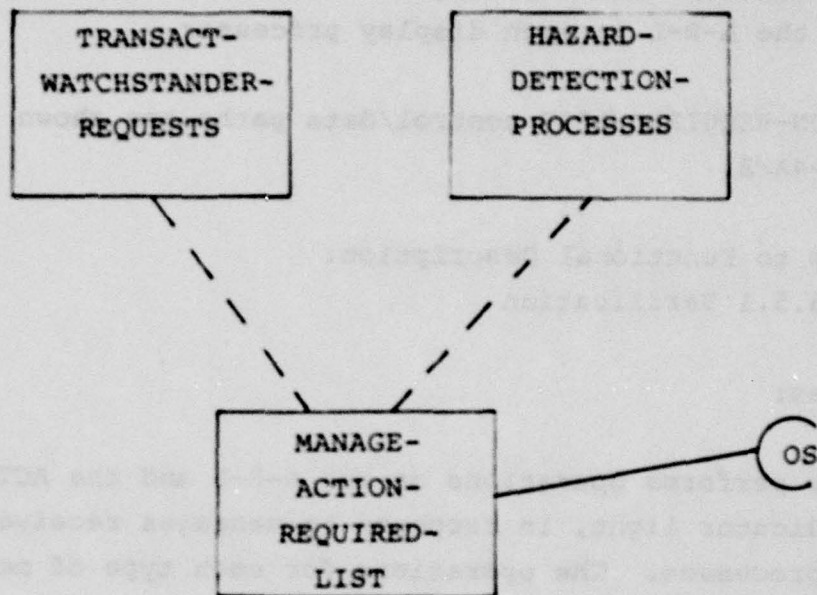


Figure 4-4A. MANAGE-ACTION-REQUIRED-LIST



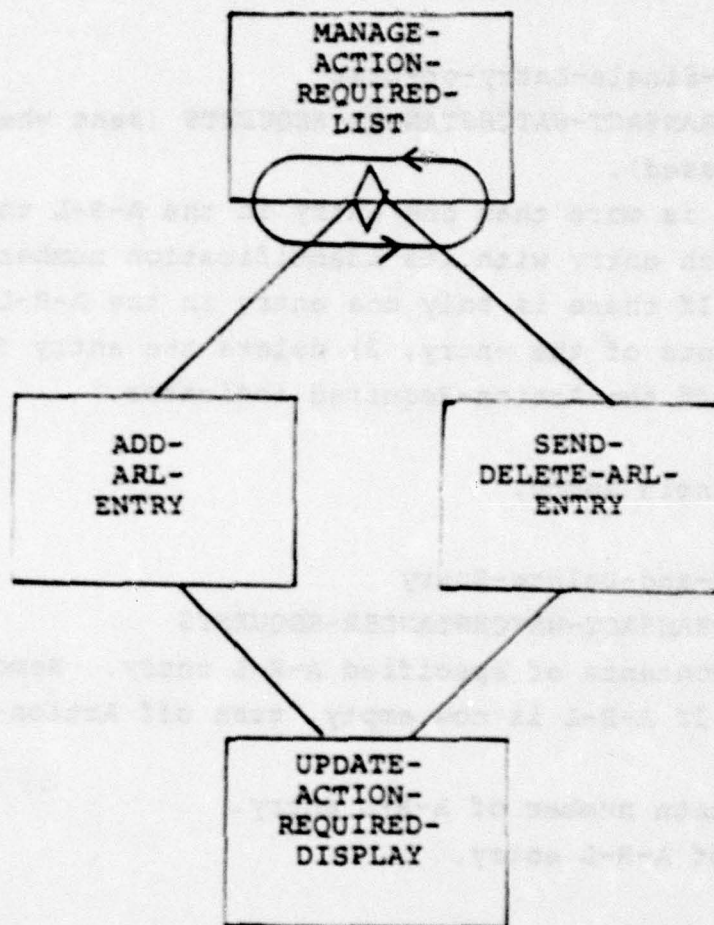


Figure 4-4B. MANAGE-ACTION-REQUIRED-LIST

INPUT: Identification number of A-R-L entry (zero implies top of list).

ANSWER: Contents of A-R-L entry.

MESSAGE TYPE: Send-Single-Entry-or-List

SENDING PROCESS: TRANSACT-WATCHSTANDER-REQUESTS (sent when ACKNOWLEDGE key pressed).

FUNCTION: If there is more than one entry in the A-R-L then return a list of each entry with its identification number and one-line summary. If there is only one entry in the A-R-L then 1) return the contents of the entry, 2) delete the entry from the A-R-L and 3) turn off the Action-Required indicator.

INPUT: None

ANSWER: List or single entry.

MESSAGE TYPE: Send-and-Delete-Entry

SENDING PROCESS: TRANSACT-WATCHSTANDER-REQUESTS

FUNCTION: Return contents of specified A-R-L entry. Remove entry from A-R-L. If A-R-L is now empty, turn off Action-Required indicator.

INPUT: Identification number of A-R-L entry.

ANSWER: Contents of A-R-L entry.

For each type of message received, the appropriate procedure is called to perform the requested function. This process coordinates (via messages) with the ACKNOWLEDGE procedure in the TRANSACT-WATCHSTANDER-REQUESTS process.

#### Data Structures:

The major data structure is the ACTION-REQUIRED-LIST (A-R-L). This is a singly-linked queue of messages specifying problems requiring action by the watchstander. Each entry in the A-R-L contains a number identifying the message and a one-line message summarizing the problem. Where required, additional information about the problem is included in the entry.



Process    MANAGE-ACTION-REQUIRED-LIST

begin

while true do

begin

if no message is queued

then wait for a message;

case message type of

                Add-Entry-to-List:    ADD-ARL-ENTRY;

                Get-ARL-Entry:    SEND-DELETE-ARL-ENTRY;

end;

            send answer to message;

end

end

Procedure ADD-ARL-ENTRY

begin

Assign an identification number to the message;  
Create an ARL entry at the bottom of the ARL;  
Copy the ID number and the message into the new entry;  
Set the ACTION-REQUIRED indicating to blink;  
UPDATE-ACTION-REQUIRED-DISPLAY;  
Wait 5 seconds;  
Set ACTION-REQUIRED indicator to on;

end

Procedure SEND-DELETE-ARL-ENTRY

begin

if identification number in the message = 0  
then  
Set the ARL pointer to the entry at the top of the  
ARL list  
else  
Find the entry with the identification number;  
Set the ARL pointer to the entry found;  
end;  
Move the contents of the ARL entry to the answer buffer;  
Delete entry from the ARL;  
UPDATE-ACTION-REQUIRED-DISPLAY;

end

Procedure UPDATE-ACTION-REQUIRED-DISPLAY

begin

for each entry in the ARL do  
Move the ID number and the one-line summary from the  
ARL entry to the next line of the Action-Required  
display buffer;  
Request the operating system to output the Action-Required  
display screen;

end



Processes in this category receive message requests primarily from the TRANSACT-WATCHSTANDER-REQUESTS process. The supporting services performed are file and table accesses, demand functions, and simulation functions. For each major file, or set of record types, there is a corresponding access process.

Processes which need to access a file record must do so by sending a message to the corresponding access process (an exception is the SEARCH-ON-KEY process which directly reads an entire file sequentially).

Where the required information for demand functions is readily available in the display station processor, these functions are performed by the TRANSACT-WATCHSTANDER-REQUESTS process. For other demand functions the TRANSACT-WATCHSTANDER-REQUESTS process sends messages to support processes in the main processor. These processes include GET-LOCAL-TRAFFIC, DETERMINE-ENCOUNTERS, ROUTE-SCHEDULE, and ACCESS-SYSTEM-PARAMETERS.

Simulation functions are performed by the SET-UP-SCENARIO process and the PLAYBACK-SCENARIO process. These processes reside in the main processor. They are invoked by messages from the TRANSACT-WATCHSTANDER-REQUESTS process at the Watch Supervisor's display station. SET-UP-SCENARIO is used to create and edit a scenario. PLAYBACK-SCENARIO is used to control the playback of a scenario.

## 5.1 VESSEL FILE ACCESS

This section describes the data structures associated with the Vessel File and the ACCESS-VESSEL-FILE process which handles all requests for access to the file. This process supports the ENTER VESSEL, MODIFY VESSEL, DELETE VESSEL, and DISPLAY VESSEL functions performed by the TRANSACT-WATCHSTANDER-REQUESTS process. In addition it supports other processes which may access the vessel file, such as SEARCH-ON-KEY. The ACCESS-VESSEL-FILE process is reentrant, allowing several invocations of the process to be concurrently processing vessel file access requests.

Relationship to Functional Description:

Appendix 8 - 5.5.1 Verification

Changes from Appendix 8: None

Frequency of Use: (Iterations per second)

	Class C	Class B	Class A
ENTER VESSEL			
Expected:	.002	.0003	.0002
Peak:	.004	.0012	.0008
MODIFY VESSEL			
Expected:	.0002	.00003	.00002
Peak:	.0008	.00012	.00008
DELETE VESSEL			
Expected	.002	.0003	.0002
Peak:	.008	.0012	.0008
DISPLAY VESSEL			
Expected:*	.12	.01	.005
Peak:	1	1	1

\*Assuming each watchstander looks at the vessel record for each vessel in the waterway once per day.



AD-A078 937

INTERNATIONAL COMPUTING CO BETHESDA MD  
VESSEL TRAFFIC SERVICES PROCESSING/DISPLAY SUBSYSTEM SOFTWARE R--ETC(U)  
SEP 79 C C HENSON , R S GRAHAM , B A MCINTOSH DOT-C6-81-78-1833

F/G 15/5

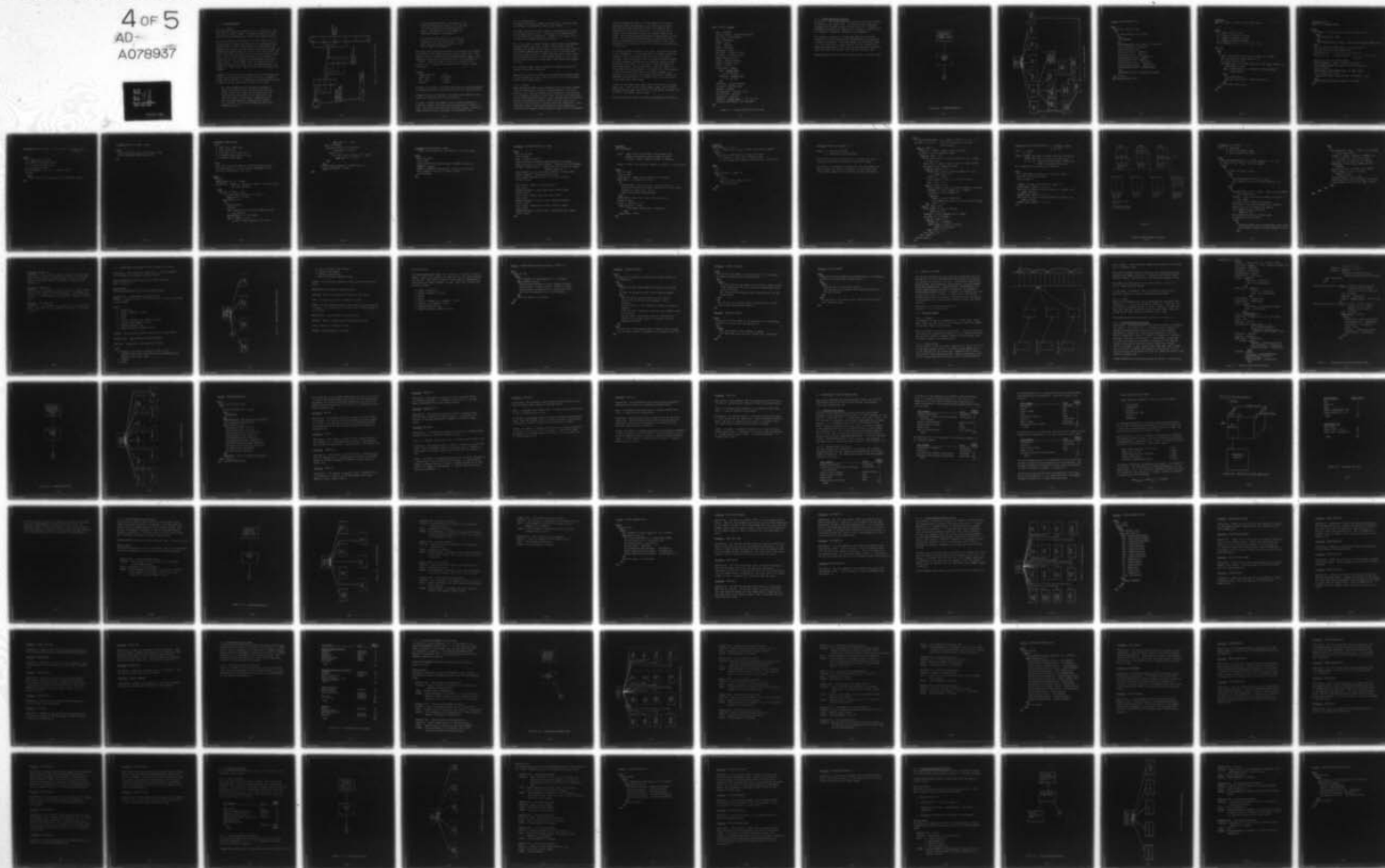
USCG -D-73-79

NL

UNCLASSIFIED

4 OF 5

AD-  
A078937





NATIONAL BUREAU OF STANDARDS  
MICROCOPY RESOLUTION TEST CHART



### 5.1.1 Data Structures

#### 5.1.1.1 Indices

The Vessel File may be accessed by one of three keys: name, call sign or Lloyd's number. Searches on any one of these keys will be performed using a three level index structure.

Figure 5-1 shows the organization for these types of files. At each level of index, the keys will be in ascending alphanumeric order. At all levels except the lowest level, the entries will indicate the first key in that block of keys. By comparing successive pairs of entries at the first level, the block containing the next (or last) index level can be identified. For all levels except the lowest, each entry will contain the key and a pointer to the beginning of the next block of keys. At the lowest level, the index entry will contain the key and the location of the desired record. An example of a three level index structure will illustrate the concept.

Suppose we have a vessel file consisting of 1000 records, and we want to set up an index structure using the (alphabetic) vessel name as the key. An example of the methods which could be used to set up a three level index is as follows:

- . The vessel names would be arranged in alphabetic order. The vessels could then be divided into ten groups of 100. Ten index entries could be set up using the key (and pointer to the next level) for the first vessel in each group. Determining that a given vessel name lies alphabetically between key  $n$  and key  $n+1$  would narrow down the next block examined to block  $n$ .

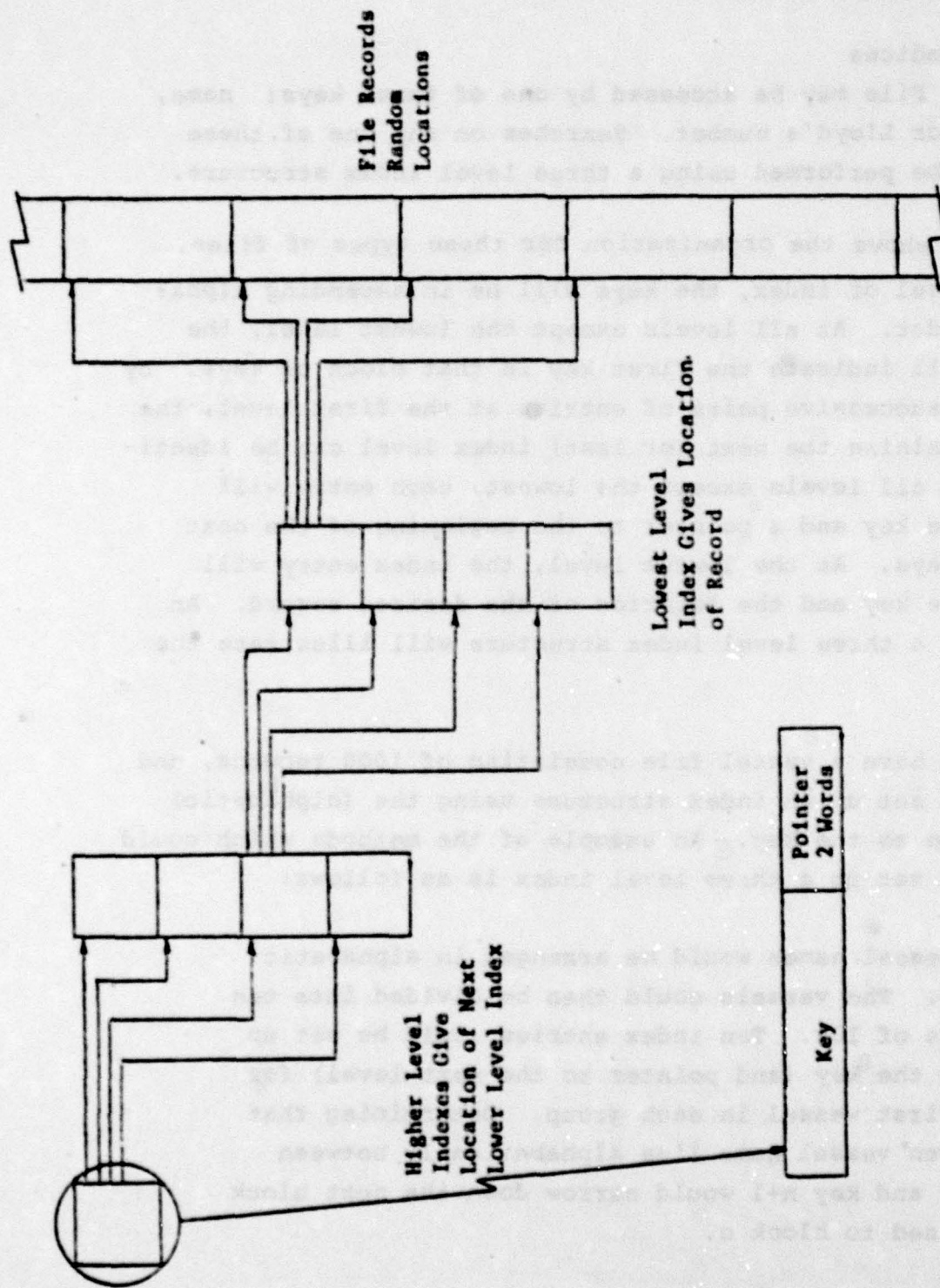


Figure 5-1 Multi-Level Index Structure



- . At the second index level, the groups of 100 could be divided into blocks of 10, with each of the ten entries set up and searched as above. This reduces to 10 the number of remaining vessels to be examined.
- . At the third index level, up to 10 keys are examined to find the vessel name. At that point the entry contains a pointer to the actual vessel record desired.

The actual size of the blocks used for the indices will depend on the size of the Vessel File and the disk sector size. The number of keys in a block should be approximately equal to the cube root of the size of the Vessel File (in the example  $10 = \sqrt[3]{1000}$ ). Since the lower levels of the indices will be stored on disc, the block size should be an integral number of sectors.

Example:

Vessel File size	10,000
Sector size	512 bytes
Key: name	25 bytes
Pointer:	4 bytes

An entry in the block is 29 bytes and there must be approximately 22 entries per block. Thus, a block must be at least 648 bytes.

Because a sector is 512 bytes, two sectors are needed for each block and 35 entries will fit in one block.

In order to reduce the number of disc accesses necessary to retrieve a record, the first level of each index is stored in core. This first level must also be stored on disc so that the index structure may be recovered if the processor fails.

#### 5.1.1.2 Record Links

The vessel records are linked in several ways. The links make it possible to find space to enter a new record.

The blank records are singly linked by the forward link field of the empty vessel record. A variable called BLANKS points to the beginning of linked list. BLANKS will be equal to zero, if there are no blank records in the file.

All of the inactive vessel records (those without corresponding passage records) are doubly linked (using forward-link and backward-link fields) in the order in which they became inactive. The variable NEWEST INACTIVE points to the record which most recently became inactive. OLDEST INACTIVE points to the record which has been inactive for the longest period of time. This is the record which will be deleted if the Vessel File becomes too full.

The variables BLANK, OLDEST INACTIVE and NEWEST INACTIVE are stored in core and on the disc.

Each active record is linked to the corresponding passage record by the forward-link field. There is also a link in the passage record to the vessel record.

#### 5.1.1.3 Locks

When creating a vessel record, modification of all three indexes could be necessary if the entry requires addition to a block which is full. This is because we are assuming a fixed maximum for the number of entries in each block, so that the index blocks do not become disproportionate in size, thereby causing potentially drastic differences in response times for data access, as a function of the location of an entry within an index block. When deleting a vessel, similar modifications to the indexes are required if the deleted vessel causes an index block to be emptied.



Serious problems may occur if two processes are allowed to access an index simultaneously. For example, one process may read the first and second levels of an index to get a pointer to a third level index block. If the third level block is full and another process is allowed access to the index, this second process may attempt to add an entry to this block. If the second process splits the third level block into two parts before the first process reads it, the first process will not be able to find the entries in the second half of the block.

To prevent multiple accesses to an index, each index will have a lock (semaphore). If a process wants to access a record or change an index, the process must wait until the index is unlocked. Then the process locks the index to prevent access by other processes and modifies the index or reads the first second and third levels of the index to find the address of the vessel record. Then the process unlocks the index to allow access by other processes. Note that it is not necessary to read the vessel record while the index is locked. The location of the record in the Vessel File does not change with additions and deletions to the Vessel File. Only the pointer in the index levels may change.

VN-LOCK is the vessel name index lock; C-LOCK is the call sign index lock; L-LOCK is the Lloyd's number index lock; B-LOCK is the blank record chain lock; NI-LOCK is the newest inactive; and OI-LOCK is the oldest inactive lock.

All the Vessel File data structures are displayed in Figure 5-2.

Vessel Record = Record

```
name: nametype;
registry number: registrynumbertype;
call sign: callsigntype;
type: vesselttype;
weight: tons;
flag: flagtype;
owner: ownertype;
max-speed: knots;
max-draft: integer;/*feet*/
min-draft: integer;/*feet*/
beam: integer;/*feet*/
length: integer;/*feet*/
height: integer;/*feet*/
doctor: (yes,no);
local agent: record
    name: packed array
        (1..36) of char;
    addrphone: packed array
        (1..55) of char;
    end;
stop-time: integer seconds;
nav-equipt: navequiptypes;
nscrews: integer;
minturnradius: integer;/*feet*/
lastverified: datatype;
lastactive: datatype;
comments: packed array (1..160) of char;
forward-link,backward-link: discaddr;
end;
```

Figure 5-2. Vessel Record Data Structures



### 5.1.2 ACCESS-VESSEL-FILE Process

Description: The ACCESS-VESSEL-FILE process handles all requests for access to the vessel file. It receives such a request as a message from another process and calls one of the following procedures to execute the vessel record operation: GET-V-R, ADD-V-R, DELETE-V-R, READ-V-R, WRITE-V-R, and RELEASE-V-R.

The process is reentrant in order that there may be several invocations of the process running concurrently, each servicing a vessel file request. Since the process is input/output bound (i.e., much more time is spent waiting for disc accesses than executing instructions), one of the invocations may be executing while the others are waiting, thus reducing response time.

ACCESS-VESSEL-FILE control/data paths are shown in Figure 5-3A/B.

"ANY PROCESS  
REQUESTING  
ACCESS TO THE  
VESSEL FILE"

ACCESS-  
VESSEL-  
FILE

OS

Figure 5-3A. ACCESS-VESSEL-FILE



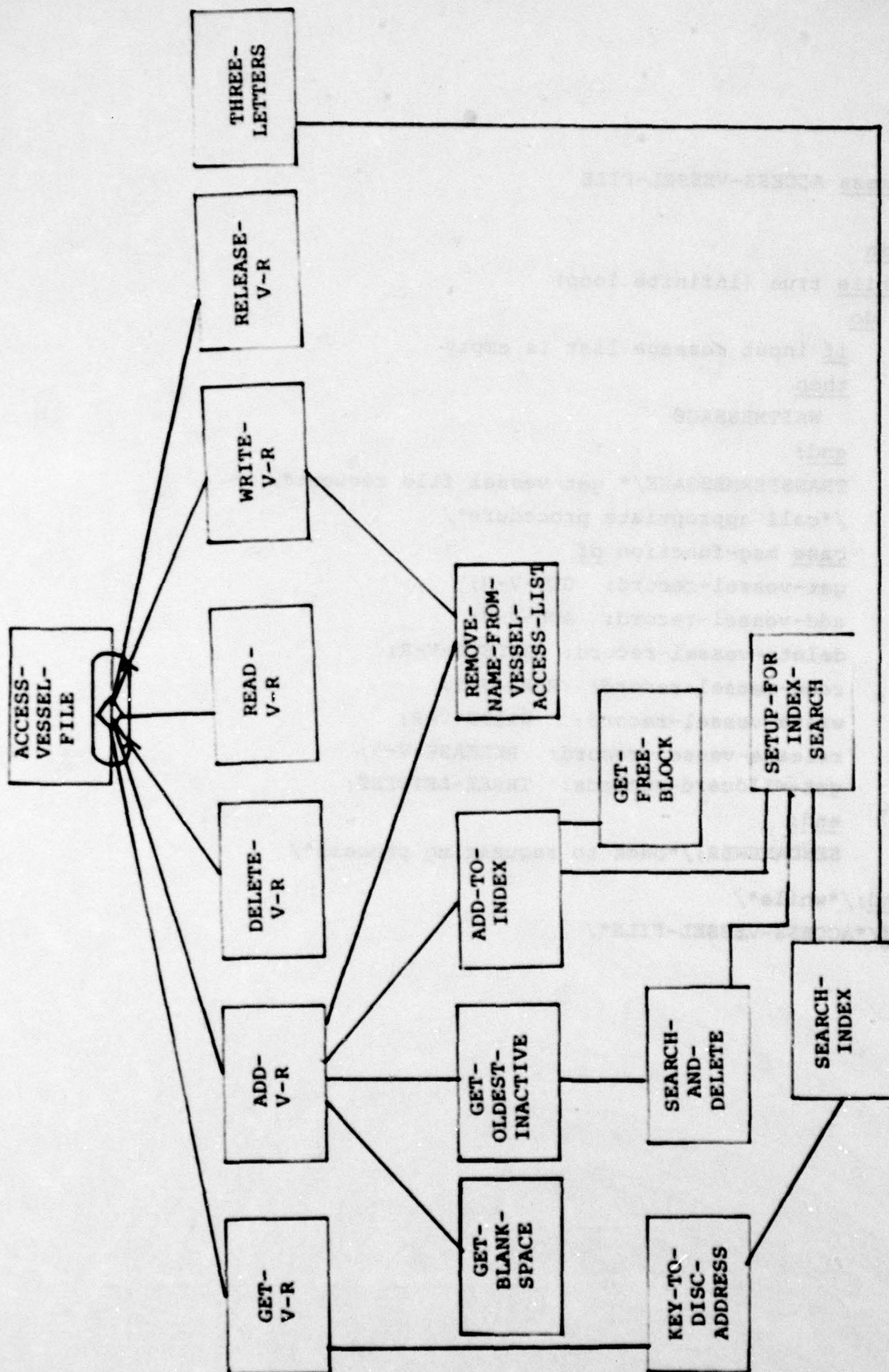


Figure 5-3B. ACCESS-VESSEL-FILE

Process ACCESS-VESSEL-FILE

begin

while true (infinite loop)

do

if input message list is empty

then

WAITMESSAGE

end;

TRANSFERMESSAGE/\* get vessel file request\*/

/\*call appropriate procedure\*/

case msg-function of

get-vessel-record: GET-V-R;

add-vessel-record: ADD-V-R;

delete-vessel-record: DELETE-V-R;

read-vessel-record: READ-V-R;

write-vessel-record: WRITE-V-R;

release-vessel-record: RELEASE-V-R;

get-wildcard-records: THREE-LETTERS;

end;

SENDANSWER;/\*back to requesting process\*/

end;/\*while\*/

end/\*ACCESS-VESSEL-FILE\*/



# Procedure

GET-V-R (index, K, access, found, ADDR, record)

## begin

B: = upper level of index;

N: = number of entries in block;

S: = number of sectors to store;

L: = number of levels of index;

KEY-TO-DISCADDR (B, K, N, S, L, ADDR, found);

## if found

### then

#### do

Read 1 record from disc starting at ADDR into RECORD;

Wait for access to Vessel Access List;

Lock Vessel Access List;

if RECORD.name is on Vessel Access List and Vessel Access List  
(name).delete = true

then /\*vessel record is in process of being deleted\*/  
found:=false;

#### else

if access#read-only

then add record.name to Vessel Access List

end;

Unlock Vessel Access List;

end;

end.

Procedure ADD-V-R

Input: VR - Vessel Record

begin

if BLANKS  $\neq$  ^ /\*there is empty record in Vessel File\*/

then

GET-BLANK-SPACE (ADDR):

else

GET-OLDEST-INACTIVE (ADDR): /\*use oldest inactive vessel list\*/

end;

Wait for NI-LOCK; /\*Add record to inactive chain\*/

Lock NI-LOCK; /\*Set name index lock\*/

VR.backward-link: = ^; /\*put VR on top of list\*/

VR.forward-link: = NEWEST-INACTIVE; /\*link to previous top of  
list \*/

Write VR on disc at location ADDR;

NEWEST-INACTIVE: = ADDR; /\*update core value\*/

Write NEWEST-INACTIVE on disc at its fixed location;

Unlock NI-LOCK;

ADD-TO-INDEX (VESSEL-NAME-INDEX, VR. NAME, ADDR);

if VR.CALLSIGN not blank

then ADD-TO-INDEX (REGISTRYINDEX, VR.REGISTRY, ADDR).

end;

REMOVE-NAME-FROM-VESSEL-ACCESS-LIST (VR.name);

end



Procedure READ-V-R (index, K, found, record) /\*without intent  
to modify\*/

begin

B: = upper level of index

N: = number of entries in block

S: = # sector to store block

L: = # levels of index

KEY-TO-DISCADDR (B, K, N, S, L, ADDR, found)

if found

then

Read 1 record from disc starting at ADDR into record;

end.

Procedure WRITE-V-R (ADDR, record)

begin

Write record onto disc at disc address ADDR;

REMOVE-NAME-FROM-VESSEL-ACCESS-LIST;

end



### Procedure THREE-LETTERS

#### Input:

B, upper level index block  
K, the 3 letter key  
N, # entries in the index block  
S, # sectors used to store a block  
L, # levels in the index

#### Output:

LIST, the list of all keys in a file which begin with the three letters in the key and the addresses of the records corresponding to the keys.

#### begin

LIST:=∅

SEARCH-INDEX (B, K, N, FIRST);

if FIRST ≠ 0 /\*FIRST = 0 implies no names in list begin with the three letters\*/

#### then

do for J:= FIRST + 1 to N

if first three letters of B(J).key = K

then add B(J) to LIST;

for COUNT:=2 to L

#### do

OLDLIST:=LIST;

LIST:=∅;

READ S sectors from location B(FIRST).pointer  
into block B;

SEARCH-INDEX (B, K, N, FIRST);

for J:= FIRST + 1 to N

if first three letters of B(J).key = K

```

        then add B(J) to LIST;
    for X in OLDLIST
        do Read S sectors from location
            x.pointer into block C;
        for J=1 to N
            if first three letters of C(J).key=K
                then add C(J) to LIST;
        end
    end
    if first three letters of B(FIRST).key = K
        then add B(FIRST) to LIST;
end

```



Procedure GET-BLANK-SPACE (ADDR);

output: returns ADDR with the address of the blank space.

begin

Wait for B-LOCK;

Lock B-LOCK;

Read a record from disc starting at BLANKS and put into  
BLANKRCD variable;

BLANKS: = BLANKRCD.forward-link; /\*update core pointer\*/

Write BLANKS on disc at its fixed location;

Unlock B-LOCK;

end.

Procedure GET-OLDEST-INACTIVE (ADDR)

begin

Wait for OI-LOCK;

LOCK OI-LOCK;

ADDR: = OLDEST-INACTIVE;

Read from disc at location OLDEST-INACTIVE into RECORD1;

Ready from disc at location RECORD1. backward-link into RECORD2.

RECORD2. forward-link:= ;/\*Break the link to previous oldest  
inactive record\*/

Write RECORD2 on disc at location RECORD1. backward-link;

OLDEST INACTIVE: = RECORD1.backward-link;

Write OLDEST-INACTIVE on disc at its fixed location;

Unlock OI-LOCK;

Wait N-LOCK; /\*Delete from name index\*/

Lock N-LOCK;

SEARCH-AND-DELETE (vessel NAME INDEX, RECORD1.NAME);

Unlock N-LOCK;

Wait C-LOCK; /\*Delete from call sign index\*/

Lock C-LOCK;

SEARCH-AND-DELETE (callsign index, RECORD1.CALLSIGN);

Unlock C-LOCK;

Wait L-LOCK; /\*Delete from Lloyd's registry index\*/

Lock L-LOCK;

SEARCH-AND-DELETE (Lloyd's index, RECORD1.registry number);

Unlock L-LOCK;

end.



Procedure

KEY-TO-DISCADDR

Input: upper level of the index to be searched, key,  
number of entries in block, number of sector  
to store a block, number of levels of indexing

Output: (found = true and disc address of record) or found = false

begin

WAIT for index

Lock on index

Index block: = upper level of index to be searched

FOR COUNT = 1 to number of levels - 1

do

SEARCH-INDEX (index block, key, no-of-entries, J);

Read from the disc the number of sectors to store a block,  
starting at indexblock(J).pointer

index block:=block read from disc.

end

SEARCH-INDEX (index block, key, no-of-entries, J);

UNLOCK-ON index;

if key = indexblock(J).key

then do found: = true;

disc addr:= index block (J).pointer;

end

else found: = false

end

Procedure

SEARCH-INDEX (B, K, N, J)

Input: index block B, key K, number of entries in block-N

Output:

The entry, J, which points to next level block

i.e.,  $\text{BLOCK}(J) \cdot \text{KEY} \leq \text{KEY} < \text{BLOCK}(J+1) \cdot \text{KEY}$

or  $J=0$  if input key is smaller than any entry in the block

begin

J:=0;

L:=1;

while B(L).Key < K and L < N

do J:=L;

    L:=L+1;

while B(L).Key = empty and L < N

do L:=L+1; end

end;

end



Procedure SEARCH-AND-DELETE (I, K)

Input: I - name of the index  
K - key which is to be deleted.

The following routine is designed for a three-level index. A similar routine may be written for a two-level index.

The index I is searched to find the key I at the lowest level and the entry for the key is removed. If the lowest level block becomes empty the route level block must be modified.

begin

SETUP-FOR-INDEX-SEARCH (I, K, LEVEL 2, LEVEL 3, J1, J2, J3);

if LEVEL3(J3).key = K /\*if not then record has already  
been deleted\*/

then do /\*part 1\*/

LEVEL3(J3): = empty /\*remove from core copy\*/

if LEVEL 3 is now completely empty

then do /\*part 1.1\*/

Return disc block to location LEVEL2(J2).ptr to empty  
block list;

LEVEL 2(J2): = empty /\*remove from 2nd level incore\*/

if LEVEL2 is now completely empty

then do /\*part 1.1.1\*/

Return disc block at location LEVEL1(J1). ptr  
to empty block list;

LEVEL1(J1): = empty

Write LEVEL1 on disc at its fixed location

end; /\*part 1.1.1\*/

else do /\*part 1.1.2\*/

Write LEVEL2 on disc at location LEVEL1(J1).pointer;

FIRSTKEY: = first nonempty key in LEVEL2

if LEVEL1(J1).key ≠ FIRST KEY

then do

LEVEL1(J1).key = FIRST KEY;

Write LEVEL1 on disc at its fixed location; end;

end; /\*part 1.1.2\*/

else do /\*part 1.2\*/

Write LEVEL3 on disc at location

LEVEL2(J2).ptr;

FIRSTKEY: = first nonempty key in LEVEL3

if LEVEL2(J2) ≠ FIRSTKEY

then do /\*part 1.2.1\*/

LEVEL2(J2): = FIRSTKEY;

WRITE LEVEL2 on disc at location

LEVEL1(J1).pointer.

end /\*part 1.2.1\*/

end /\*part 1.2\*/

end; /\*part 1\*/

Unlock index I;

end.



Procedure SETUP-FOR-INDEX-SEARCH (I, K, LEVEL2, LEVEL3,  
J1, J2, J3)

Input: I - index

K - key

Output: LEVEL2 core copy of second level block for key K.

LEVEL3 core copy of 3 level block for key K.

J1, J2, J3: Positions within LEVEL1, LEVEL2 and

LEVEL3 corresponding to key K

The variables are illustrated in Figure 5-4.

begin

N: = the number of entries in a block for index I;

Wait for the lock for index I;

Lock on index I;

/\*LEVEL1 is upper level block for index I\*/

SEARCH-INDEX (LEVEL1, K, N, J1);

Read from the disc at location LEVEL1(J1). pointer into  
LEVEL2 variable

SEARCH-INDEX (LEVEL2, K, N, J2);

Read from the disc at location LEVEL2(J2).pointer into  
LEVEL3 variable

SEARCH-INDEX (LEVEL3, K, N, J3);

end.

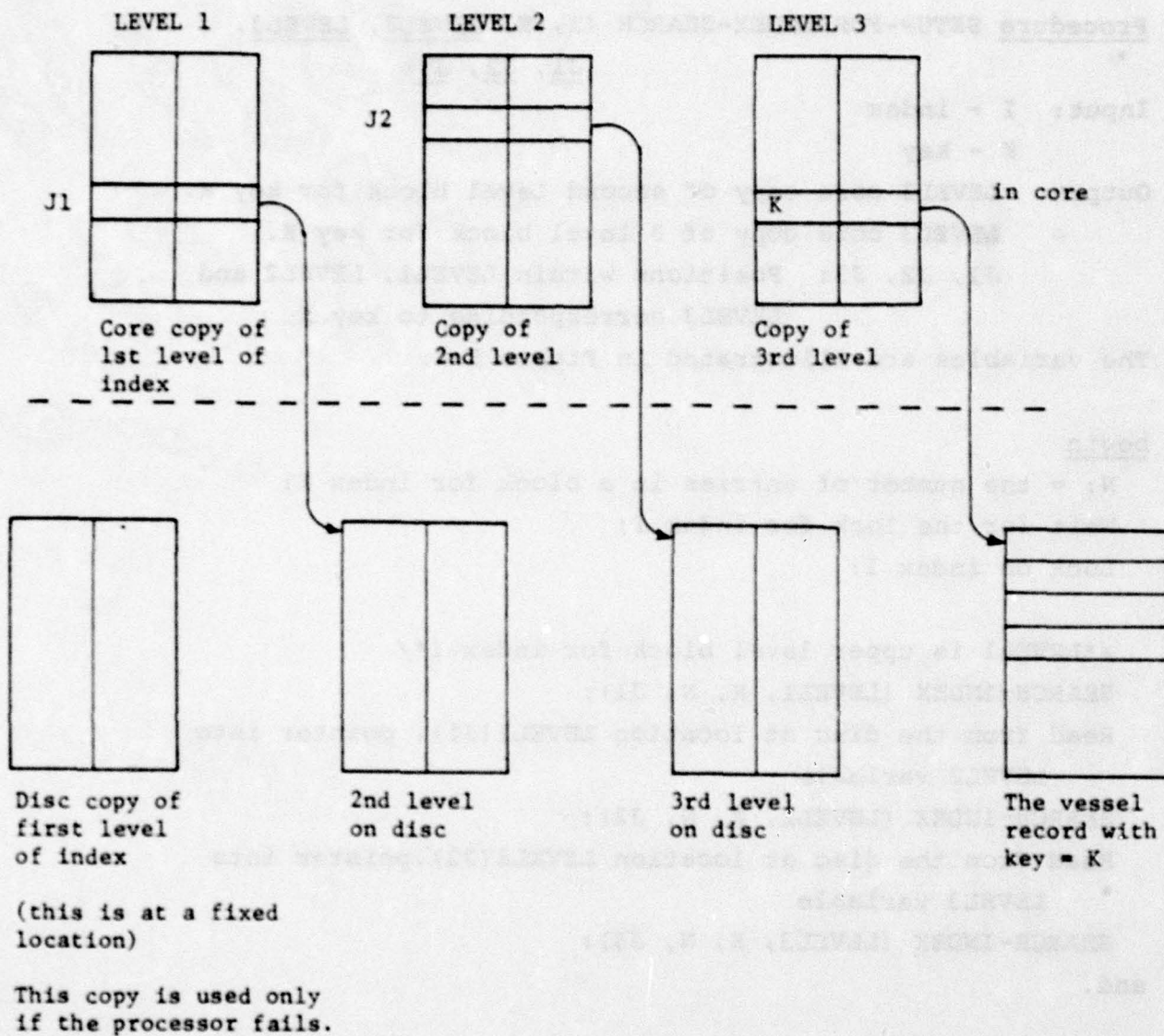


Figure 5-4



Procedure ADD-TO-INDEX (I, K, P);

Input: I, index name

K, key to be added to index

P, disc of record will key K

begin

SETUP-FOR-INDEX-SEARCH (I, K, LEVEL2, LEVEL3, J1, J2, J3);

if LEVEL3(J3)  $\neq$  K /\*then K is not in index\*/

then

do

if LEVEL3 has room in block

then

do

rearrange LEVEL3 with the K, P entry,

write LEVEL3 on disc at location LEVEL2(J2).ptr

end;

else

do

GET-FREE-BLOCK (LOC1); /\*LOC1; /\*LOC1 is disc address  
of the free block\*/

LEVEL3A: = last N/2 entries in LEVEL3 with the K, P  
entry added;

Fill last half of LEVEL3A with empties;

FIRST KEY1: = the first nonempty key of LEVEL3A;

Replace last half of LEVEL3 with empties

Write LEVEL3 on disc at location

LEVEL2(J2).ptr;

Write LEVEL3A on disc at location LOC1;

if LEVEL2 has room in block

then

do

rearrange LEVEL2 with the FIRST KEY1, LOC1 entry,

write LEVEL2 on disc at location LEVEL1(J1).ptr;

end;

```

else
do
  GET-FREE-BLOCK (LOC2); /*LOC2 is disc address
    of the free block*/
  LEVEL2A: = last N/2 entries in LEVEL2 with
    the FIRST KEY1, LOC1 entry added;
  Fill last half of LEVEL 2A with empties;
  FIRST KEY2: = the first nonempty key of
    LEVEL2A;
  Replace last half of LEVEL2 with empties
  Write LEVEL2 on disc at location
    LEVEL1(J1).ptr;
  Write LEVEL2A on disc at LOC2;
  Rearrange LEVEL1 with FIRST KEY2, LOC2 entry
  Write LEVEL1 on disc at its fixed location
end;
end;
end;
end.

```



Procedure DELETE -V-R

Description: This module is called to delete a vessel record from the vessel file. It performs this function in the same manner as the module DELETE-P-R removes a vessel from the passage file.

Procedure RELEASE-V-R

Description: This module is called to unlock a vessel record that has been locked by the GET-V-R module. It performs this function in the same manner that RELEASE-P-R unlocks a passage record.

Procedure GET-FREE-BLOCK

Description: This searches the Vessel File for an unused disc block. It returns the disc address of the block to the calling module.

### 5.1.3 MANAGE-VESSEL-INFORMATION-TABLE (MANAGE-VIT) Process

Description: This process is responsible for the maintenance of and access to the Vessel-Information-Table.

MANAGE-VESSEL-INFORMATION-TABLE control/data paths are shown in Figure 5-5.

#### Message Types:

The program processes the following message types:

MESSAGE TYPE: Create-Vessel-Information-Entry

FUNCTION: Create an entry in the VIT with the information provided in this message.

#### INPUT:

- 1) Internal ID
- 2) Vessel (external) ID code
- 3) Status
- 4) Draft
- 5) List of intended route segments (if any)
- 6) Current route segment (if any)
- 7) Tracked/Untracked flag
- 8) Hazard processing exemption flags

ANSWER: Flag indicating whether vessel entry already exists.

MESSAGE TYPE: Update-Vessel-Information-Entry

FUNCTION: Change data in an existing VIT entry.

#### INPUT:

- 1) Internal ID or Vessel (external) ID code of the following fields, any which are not to be changed should contain a null (zero) value
- 2) Status
- 3) Draft



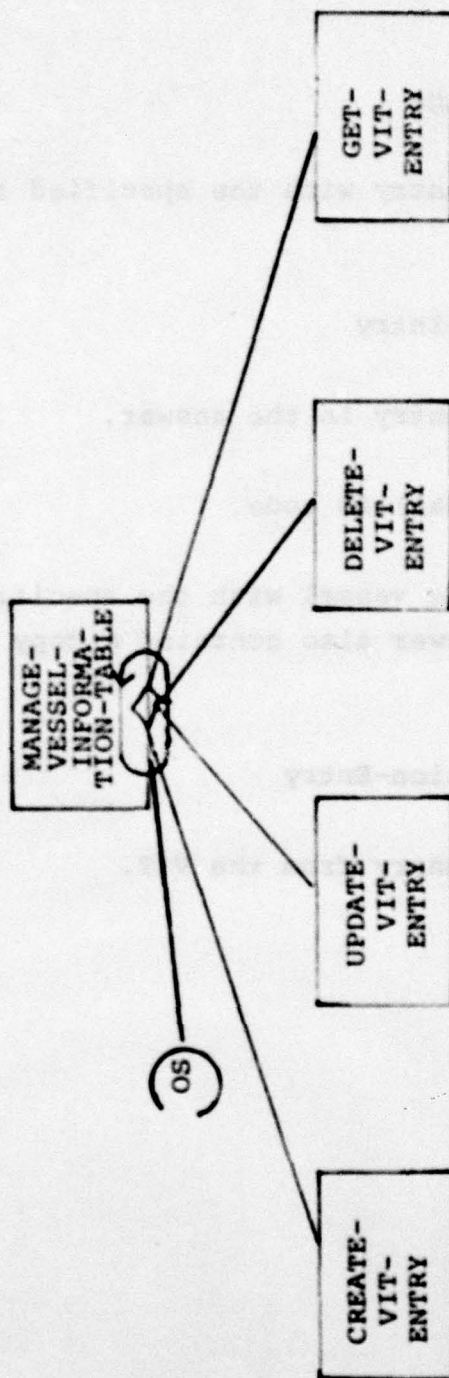


Figure 5-5. MANAGE-VESSEL-INFORMATION-TABLE

- 4) List of intended route segments
- 5) Current route segment
- 6) Tracked/untracked flag
- 7) Hazard processing exemption flags

ANSWER: Flag indicating whether an entry with the specified ID exists.

MESSAGE TYPE: Get-Vessel-Information-Entry

FUNCTION: Return the specified VIT entry in the answer.

INPUT: Internal ID or Vessel (external) ID code.

ANSWER: A flag indicating whether the vessel with the specified ID exists. If it exists then the answer also contains a copy of the VIT entry.

MESSAGE TYPE: Delete-Vessel-Information-Entry

FUNCTION: Remove the specified VIT entry from the VIT.

INPUT: Internal or external ID code.

ANSWER: Acknowledgement of deletion.



#### Data Structures:

The VESSEL-INFORMATION-TABLE (VIT) consists of an entry for each vessel. There is an internal ID index (as with the Position-Table) into the VIT. The actual location of each entry is determined by hash-coding the vessel external ID code. Each entry contains the following fields:

- 1) Vessel internal ID
- 2) Vessel (external) ID code
- 3) Status
- 4) Draft
- 5) List of intended route segments (if any)
- 6) Current route segment (if any)
- 7) Tracked/Untracked flag
- 8) Hazard processing exemption flags

Process   MANAGE-VESSEL-INFORMATION-TABLE (MANAGE-VIT)

begin

while true do

begin

if no message is queued then wait for a message;

case message type of

        Create-Vessel-Information-Entry: CREATE-VIT-ENTRY;

        Update-Vessel-Information-Entry: UPDATE-VIT-ENTRY;

        Delete-Vessel-Information-Entry: DELETE-VIT-ENTRY;

        Get-Vessel-Information-Entry: GET-VIT-ENTRY;

end;

      Send the answer to the message;

end

end



Procedure CREATE-VIT-ENTRY

begin

if there is not already an entry with the same internal ID  
or external ID

then

begin

Compute the table entry address by hashing the external  
ID;

if there is already an entry at the computed address

then

Find the first unoccupied space in the overflow  
area of the VIT and set entry address to that  
location;

Set the pointer in the internal ID index to the entry  
address;

Move the vessel information from the input message buffer  
to the entry;

Set the flag in the answer buffer to indicate that a  
vessel with the same internal or external ID is not  
already in the VIT;

end

else

Set the flag in the answer buffer to indicate that a vessel  
with the same internal or external ID is already in the VIT;

end

Procedure UPDATE-VIT-ENTRY

begin

Send for the entry based on the specified ID in the message;

if there is an entry with the specified ID

then

begin

For each field in the message which does not contain a null  
(zero) value, move the value into the corresponding field  
in the VIT entry;

Set the flag in the answer buffer indicating that the VIT  
entry exists;

end

else

Set the flag in the answer buffer indicating that the VIT  
entry exists and has been updated;

end

Procedure DELETE-VIT-ENTRY

begin

Search for the entry based on the specified ID in the message;

if there is an entry with the specified ID

then

begin

Zero the pointer in the internal ID index;

Fill the entry space with null (binary zero) characters;

end

end



Procedure GET-VIT-ENTRY

begin

Search for the entry based on the specified ID in the message;

if there is an entry with the specified ID

then

begin

Copy the entry to the answer buffer;

Set the flag in the answer buffer indicating that the

VIT entry exists;

end

else

Set the flag in the answer buffer indicating that the VIT

entry does not exist;

end

## 5.2 PASSAGE FILE ACCESS

This section describes the data structures associated with the Passage File and the ACCESS-PASSAGE-FILE process which handles all requests for access to the file. This process supports the ENTER-PASSAGE, MODIFY-PASSAGE, DELETE-PASSAGE, DISPLAY-PASSAGE, ENTER-NEW-COMMUNICATIONS, IDENTIFY-VESSEL, CHANGE-STATUS, and UPDATE-VESSEL-POSITION functions performed by the TRANSACT-WATCHSTANDER-REQUESTS process. In addition it supports any other process which needs to access the passage file. The ACCESS-PASSAGE-FILE process is reentrant, allowing several invocations of the process to be concurrently processing vessel file access requests.

Relationship to Functional Description:

Appendix 8 - 5.5.1 Verification

### 5.2.1 Data Structures

#### 5.2.1.1 Indices

The Passage File may be accessed by one of three keys: vessel name, vessel ID code (this is the 4 character code which appears on the map display) or pilot name.

Refer to Figure 5-6 for index organization. This is the same as the vessel file organization (Section 5.1.1.1) except that there are only two levels of indexing instead of three because there are fewer records in the passage file.

#### 5.2.1.2 Record Links

Unused passage records are singly linked by the vessel link field of the empty passage record. A variable called BLANKS points to the beginning of the linked list. BLANKS is set to zero if there are no unused passage records. There is similarly a list of free communications records and a list of unused reporting



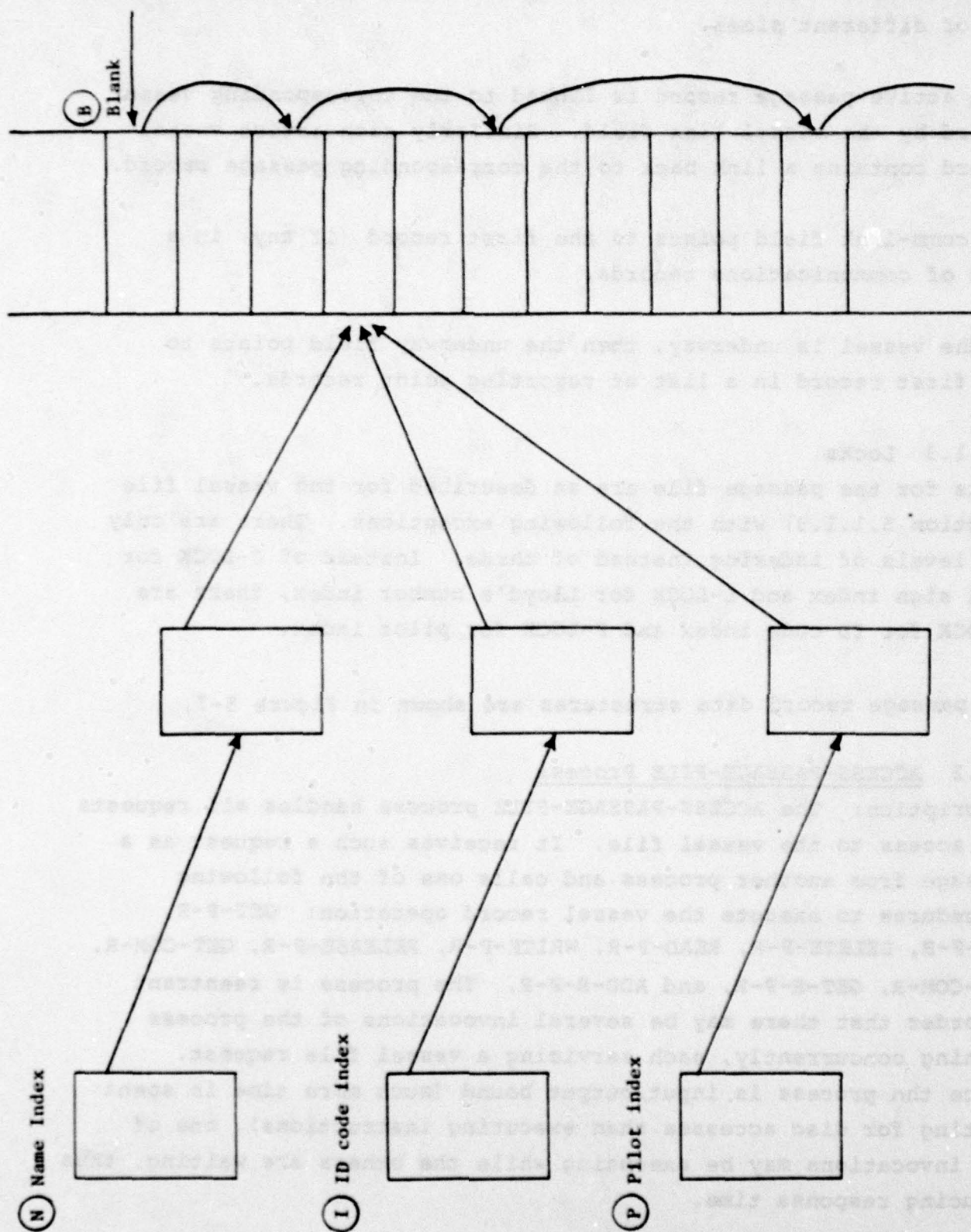


Figure 5-6 Passage File Index Organization

point records. There are three separate lists because the records are of different sizes.

Each active passage record is linked to the corresponding vessel record by the vessel link field. Similarly each active vessel record contains a link back to the corresponding passage record.

The comm-link field points to the first record (if any) in a list of communications records.

If the vessel is underway, then the underway field points to the first record in a list of reporting point records.

#### 5.2.1.3 Locks

Locks for the passage file are as described for the vessel file (Section 5.1.1.3) with the following exceptions. There are only two levels of indexing instead of three. Instead of C-LOCK for call sign index and L-LOCK for Lloyd's number index, there are I-LOCK for ID code index and P-LOCK for pilot index.

The passage record data structures are shown in Figure 5-7.

#### 5.2.2 ACCESS-PASSAGE-FILE Process

Description: The ACCESS-PASSAGE-FILE process handles all requests for access to the vessel file. It receives such a request as a message from another process and calls one of the following procedures to execute the vessel record operation: GET-P-R, ADD-P-R, DELETE-P-R, READ-P-R, WRITE-P-R, RELEASE-P-R, GET-COM-R, ADD-COM-R, GET-R-P-R, and ADD-R-P-R. The process is reentrant in order that there may be several invocations of the process running concurrently, each servicing a vessel file request. Since the process is input/output bound (much more time is spent waiting for disc accesses than executing instructions), one of the invocations may be executing while the others are waiting, thus reducing response time.

ACCESS-PASSAGE-FILE control/data paths are shown in Figure 5-8A/B.



Passagerecord = record

```
Vesselid: vesselidtype; /*4 character code*/
Internalid: integer (1..7999) /*internal passage ID*/
Vesselname: nametype;
Typevessel: vesseltype;
Passagestatus: status;
Presentsector: sectortype;
Origin: record of
    location: coordinate;
    name: positionnametype;
    end;
Destination: record of
    location: coordinate;
    name: positionnametype;
    end;
Entrytimedate: timedatetype;
Exittimedate: timedatetype;
pilot: record
    designation: pilotdesignationtype
    name: pilotnametype;
    end
cargo: (cargotypes);
bargecount: integer; /*for tug or tow boat*/
draft: real;
laneflag: (yes,no);
intendedroute: packed array (1..100)
    of routesegmentnametype;
    /*temporary/permanent flag
    encoded in route segment name*/
vessellink: discaddr;
commlink: discaddr;
case status: statustype of
    docked: record
        dockname: dockdesignationtype
        arrivaltimedate: timedatetype
        departuretimedate: timedatetype
        end;
    anchored: record
        anchorage: anchoragenametype;
        anchorage: coordinate;
        swing: integer;
        anchoredatime: timedatetype;
        end;
```

Figure 5-7. Passage Record Data Structures

```

imminent: ( ) ;
underway: discaddr
           /*pointer to first disc block
           containing reporting point history*/
end; /*end case*/
end;

communications records = record /*linked in reverse order of
                                timedate with most recent
                                record the first on the list*/
link: discaddr;
timedate: timedatetype;
summary: packed array (1..160) of char
end;

reporting point record = record /*linked in reverse order of
                                passage with most recent
                                reporting point first*/
link: discaddr;
packed array (0..50) of
record
    currentpoint: reportingpointtype;
    currentpointtime: timedatetype;
    routesegment: routesegment
    speed: knots;
    nextpoint: reportingpointtype
    nextpointtime: timedatetype;
end;
end;

```

Figure 5- 7 Passage Record Data Structures (Cont.)



"ANY PROCESS  
REQUESTING  
ACCESS TO THE  
PASSAGE  
FILE

ACCESS-  
PASSAGE-  
FILE

OS

Figure 5-8A. ACCESS-PASSAGE-FILE

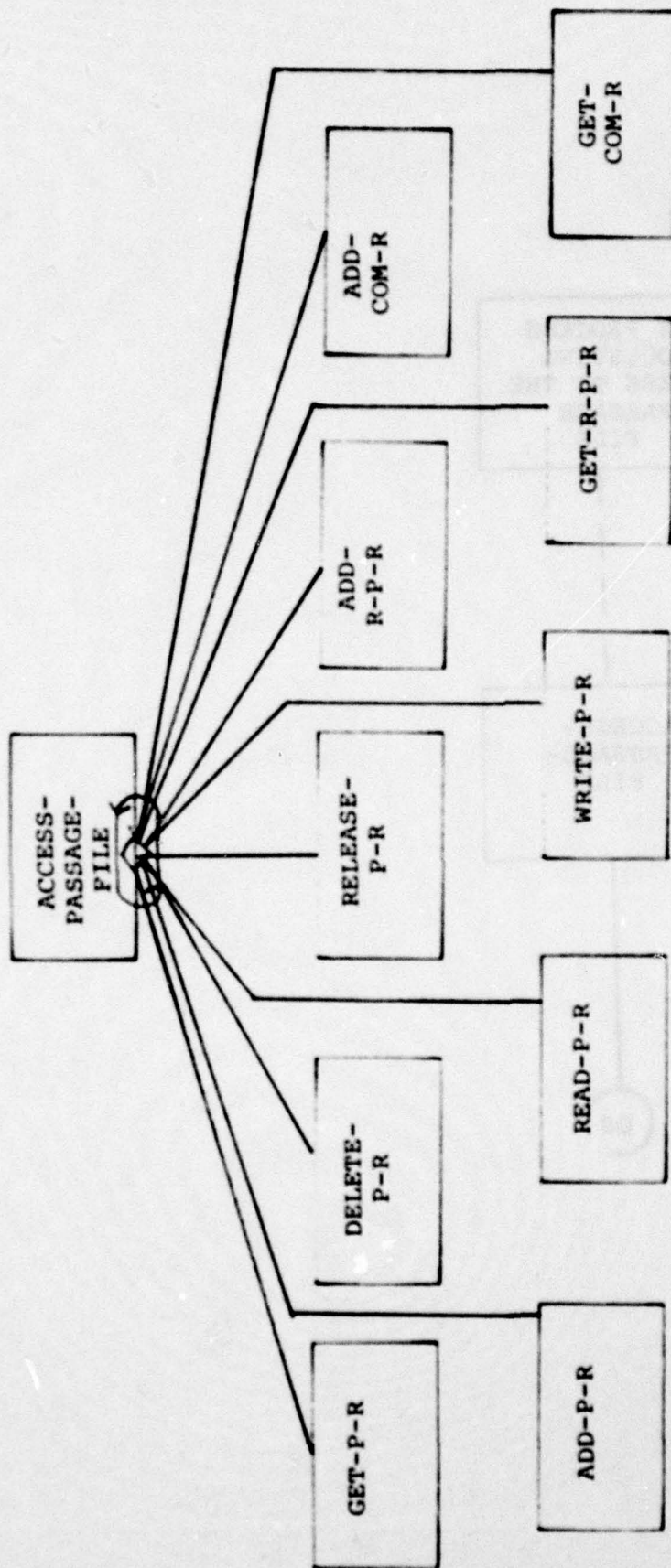


Figure 5-8B. ACCESS-PASSAGE-FILE



Process ACCESS-PASSAGE-FILE

begin

while true (infinite loop)

do

if input message list is empty

then

WAITMESSAGE

end;

TRANSFERMESSAGE /\*get passage file request\*/

/\*call appropriate procedure\*/

case msg-function of

get-passage-record: GET-P-R;

add-passage-record: ADD-P-R;

delete-passage-record: DELETE-P-R;

read-passage-record: READ-P-R;

write-passage-record: WRITE-P-R;

release-passage-record: RELEASE-P-R;

get-reporting-point-record: GET-R-P-R;

add-reporting-point-record: ADD-R-P-R;

get-comm-records: GET-COM-R;

add-comm-record: ADD-COM-R;

end;

SENDANSWER; /\*back to requesting process\*/

end; /\*while\*/

end; /\*ACCESS-PASSAGE-FILE\*/

The procedures for the ACCESS-PASSAGE-FILE process are analogous to those of the ACCESS-VESSEL-FILE process but with additional procedures for locating and adding communications records and reporting point records which are linked to the passage record. These procedures are:

Procedure GET-P-R

Description: This module locates a passage record and returns its address. It locks out the record preventing other access to it until it is unlocked by calling the RELEASE-P-R module. This module should be called before modifying or detecting a passage record.

Procedure ADD-P-R

Description: This module is called to add a passage record to the passage file. It performs this function in the same way that the ADD-V-R module adds a vessel record to the vessel file.

Procedure DELETE-P-R

Description: This module is called to delete a passage record from the passage file. It performs this function in the same manner as the module DELETE-V-R removes a vessel from the vessel file.

Procedure READ-P-R

Description: This module is called to read a passage record. It performs this function in the same manner that module READ-V-R reads a vessel record.



Procedure WRITE-P-R

Description: This module is called to write a passage record at a specified address. It performs this function in the same manner that WRITE-V-R writes a vessel record.

Procedure RELEASE-P-R

Description: This module is called to unlock a passage record that has been locked by the GET-P-R module. It performs this function in the same manner that RELEASE-V-R unlocks a vessel record.

Procedure GET-COM-R

Description: This procedure locates and returns all communications records linked to a specified passage record.

Input: 1) Passage record index type. 2) Passage record index value.

Processing: The passage record is found by searching the specified index. When the passage record is found the chain of communications record is followed, adding each record and its address to a list.

Output: An answer is composed containing the following information:  
1) Status set to one of the following values: a) comm record(s) found; b) passage record not found; c) passage record has no comm records, 2) Number of comm records, 3) list of comm records and their disc addresses.

Procedure ADD-COM-R

Description: This procedure adds a communications record onto the chain of comm records linked to a passage record.

Input: 1) passage record index type; 2) passage record index value;  
3) copy of communications record.

Processing: The passage record is found by searching the specified index. When the passage record is found the chain of comm records is followed to the end, where the new comm record is added.

Output: An answer is composed containing the following information:

1) status is set to one of the following: a) comm record added successfully, b) passage record not found; c) invalid index type; d) invalid index value.



Procedure GET-R-P-R

Description: This procedure locates and returns a designated reporting point record for a specified passage record.

Input: 1) passage record index type; 2) passage record index value; 3) reporting point designation.

Processing: The specified index is searched for the specified index value. When the passage record is located the chain of reporting point records is searched for the record with the specified reporting point designation.

Output: An answer is composed containing the following information:  
1) status equals one of the following: a) reporting point record found; b) passage record not found; c) reporting point record not found. 2) Disc address of reporting point record. 3) Copy of reporting point record.

Procedure ADD-R-P-R

Description: This procedure adds a reporting point record onto the chain of reporting point records linked to a passage record.

Input: 1) Passage record index type. 2) passage record index value. 3) Copy of reporting point record.

Processing: The passage record is found by searching the index to find a match with the index value. When the passage record is found, the chain of reporting point record is followed to the end, where the new reporting point record is linked.

Output: An answer is composed containing one of the following status indications: 1) Reporting point record added successfully. 2) Passage record doesn't exist. 3) Invalid index type. 4) Invalid index value.



### 5.3 MISCELLANEOUS FILE AND TABLE ACCESS

This section describes the processes which access the files and tables which contain the data on the waterway, environment, notices and watchstander IDs.

#### 5.3.1 Waterway File Access

Included in this section are the data structures and access processes for the following record types: 1) route segments, 2) way-points, 3) docks and piers, 4) navigational aids, and 5) cells. With the exception of temporary route segments associated with a vessel passage, it is recommended that none of the waterway data be added, modified or deleted while the system is active. Changes to the waterway data will be performed off-line to avoid unnecessary complications. Since these updates are performed off-line, one at a time, there is only one non-reentrant process needed to perform the functions, UPDATE-WATERWAY-FILES. An on-line reentrant process, ACCESS-WATERWAY-DATA, would be used to read waterway data and to enter, modify and delete temporary route segments only.

##### 5.3.1.1 Waterway Data Structures

The Route-Segment-File consists of a sequential file of Route-Segment-Records which are ordered lexicographically by the route segment designation. The content and format of the Route-Segment-Record is as follows:

DATA ELEMENT	UNITS	LENGTH (BYTES)
Segment Designation	Characters	6
Coordinates of End Points of Straight Line Portions	Coordinates	80
Length of Route Segment	Nautical Miles	4
Minimum Depth at MLLW	Feet	1
Speed Limit	Knots	1
Minimum Width of Channel	Feet	2
TOTAL		94

In addition to the permanent route segments described above, there is a Temporary-Route-Segment-File. This also is a sequential file with records ordered lexicographically by segment designation. The Temporary-Route-Segment-Record format is as follows:

<u>DATA ELEMENT</u>	<u>UNITS</u>	<u>LENGTH (BYTES)</u>
Segment Designation	Characters	6
Coordinates of End Points of Straight Line Portions	Coordinates	80
Length of Route Segment	Nautical Miles	4
Minimum Depth at MLLW	Feet	1
Speed Limit	Knots	1
Link to Passage Record	Disc Address	<u>4</u>
TOTAL		96

The Waypoint-File is a sequential file containing records of the following format:

<u>DATA ELEMENT</u>	<u>UNITS</u>	<u>LENGTH (BYTES)</u>
Designation of Waypoint	Characters	6
Location	Coordinates	4
In-Bound Route Segment Designations	Characters	24
Out-Bound Route Segment Designations	Characters	<u>24</u>
TOTAL		58



The Docks-Piers-File is a sequential file with records ordered lexicographically by designation. The record format is as follows:

<u>DATA ELEMENT</u>	<u>UNITS</u>	<u>LENGTH (BYTES)</u>
Designation	Characters	14
Length	Feet	2
Depth at MLLW	Feet	1
Width	Feet	2
Facilities Available	Types	4
Location	Coordinates	4
Name of Owner	Characters	36
Address and Phone Number	Characters	<u>55</u>
TOTAL		118

The Nav aids-File is a sequential file with records ordered lexicographically by designation. The record format is as follows:

<u>DATA ELEMENT</u>	<u>UNITS</u>	<u>LENGTH (BYTES)</u>
Designation	Characters	6
Location	Coordinates	4
Watch Circle Radius	Feet	2
Type of Aid	Types	1
Adrift or Missing Processing Flag		<u>1</u>
TOTAL		14

The VTS coverage area will be subdivided into cells which will be used to describe the characteristics of the waterway. The waterway database is structured at three levels, the highest of which is resident in memory. See Figure 5-1.

At the first level the VTS area is divided into cell blocks. Each block represents a 10 x 10 matrix of cells. The memory resident descriptor is a single word containing:

. Mean lowest low water (MLLW)

. Flags describing features of the cell block including:

- Routes/lanes
- Docks/piers
- Hazards
- Navigational aids
- Sensors
- Waypoints

In a maximum coverage VTS, the memory table would represent a 40 x 40 matrix with each word describing a block of cells with a total size of approximately 10 nautical miles on a side. The Cell-File contains the cell blocks on disc.

Each cell block is a 10 x 10 matrix of individual cell descriptions which are each 8 words. Each cell represents an area that is approximately 1 nautical mile on a side. Included in the cell descriptor is the following: (See Figure 5-9A/B.)

. Mean lowest low water for the cell	1 byte
. Speed limit in cell	1 byte
. Flags describing cell features	2 bytes
. Subcell descriptor	8 bytes
. Pointer to supplementary data	4 bytes

The subcell descriptor included above contains a 4 x 4 matrix of 4 bits each. The data items are a coded representation of the MLLW within a cell. The value zero indicates a water level equal to the MLLW for the entire cell. Values 1 through 15 determine the water level by the formula:

$$MLLW_{\text{subcell}} = MLLW_{\text{cell}} + 2 \times \text{VALUE}$$



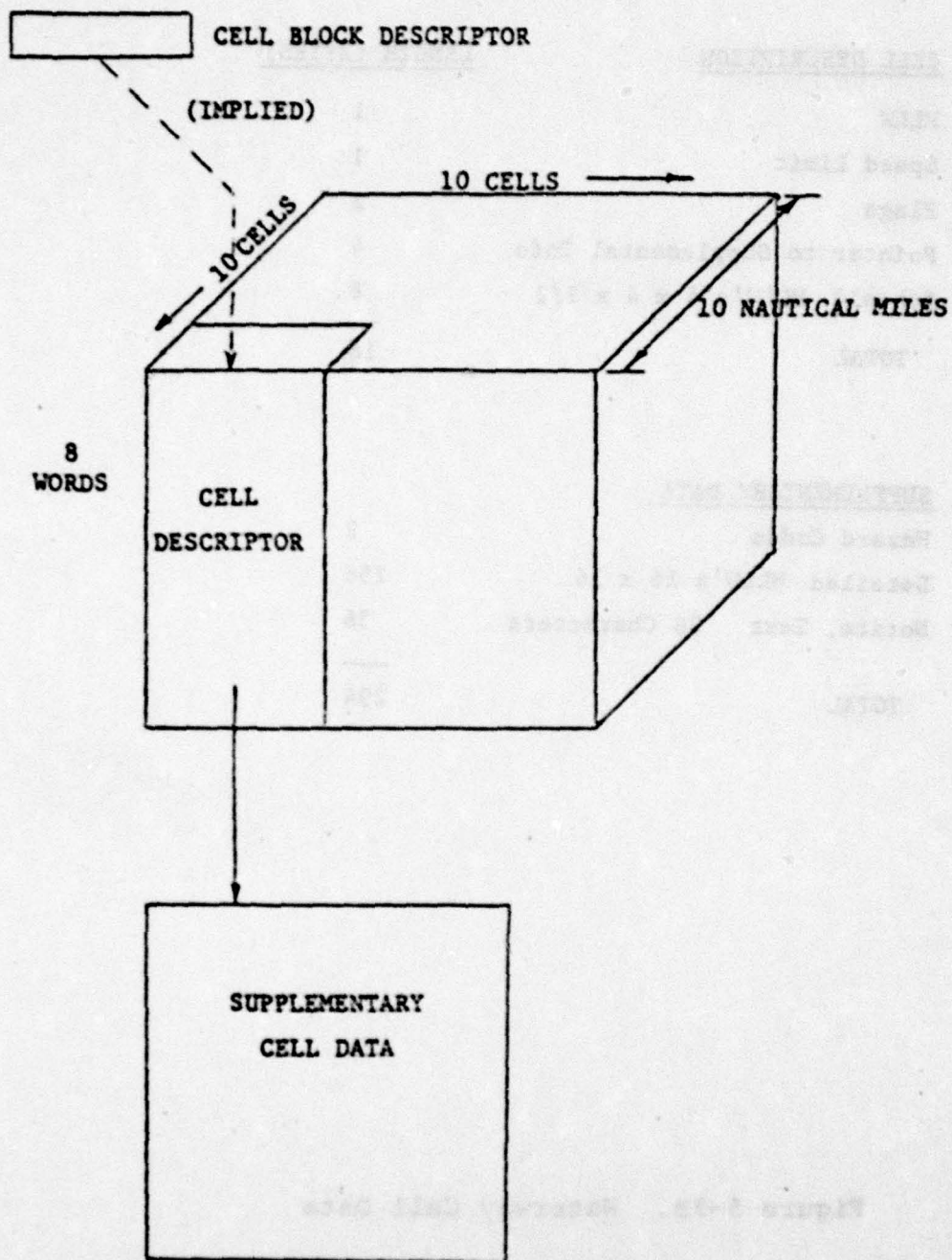


Figure 5-9A. Waterway Data Base Description

<u>CELL DESCRIPTION</u>	<u>LENGTH (BYTES)</u>
MLLW	1
Speed Limit	1
Flags	2
Pointer to Supplemental Info	4
Subcell MLLW's 4 x 4 x 1/2	8
	<hr/>
TOTAL	16

<u>SUPPLEMENTARY DATA</u>	
Hazard Codes	2
Detailed MLLW's 16 x 16	256
Notice, Text 36 Characters	36
	<hr/>
TOTAL	294

Figure 5-9B. Waterway Cell Data



The cell descriptor contains the MLLW for each area of the waterway down to approximately 1/4 nautical mile on a side. If additional detail is needed for MLLWs or if the flag word indicates the presence of hazards or other features requiring elaboration, a detail block will exist containing the required data.

#### 5.3.1.2 ACCESS-WATERWAY-DATA Process

This process is used to obtain data from route segment, cell, navaid or dock/pier records. In addition, it may be called to enter, modify, or delete temporary route segment records. The process is reentrant; therefore, it may be processing several requests concurrently. Since the process is input/output bound, this increases efficiency and reduces response time.

ACCESS-WATERWAY-DATA control/data paths are shown in Figure 5-10 A/B.

#### Message Types:

The process consists of a set of procedures, each of which performs the function requested by one of the following message types.

MESSAGE TYPE: Get-Route-Segment-Record

FUNCTION: Read and return the contents of the specified route segment record.

INPUT: Designation of route segment.

ANSWER: Acknowledgement of message, flag indicating whether route segment record exists, and if found, the contents of the route segment record.



"ANY PROCESS  
REQUESTING  
ACCESS TO THE  
WATERWAY FILES"

ACCESS-  
WATERWAY-  
DATA

OS

FIGURE 5-10A. ACCESS-WATERWAY-DATA

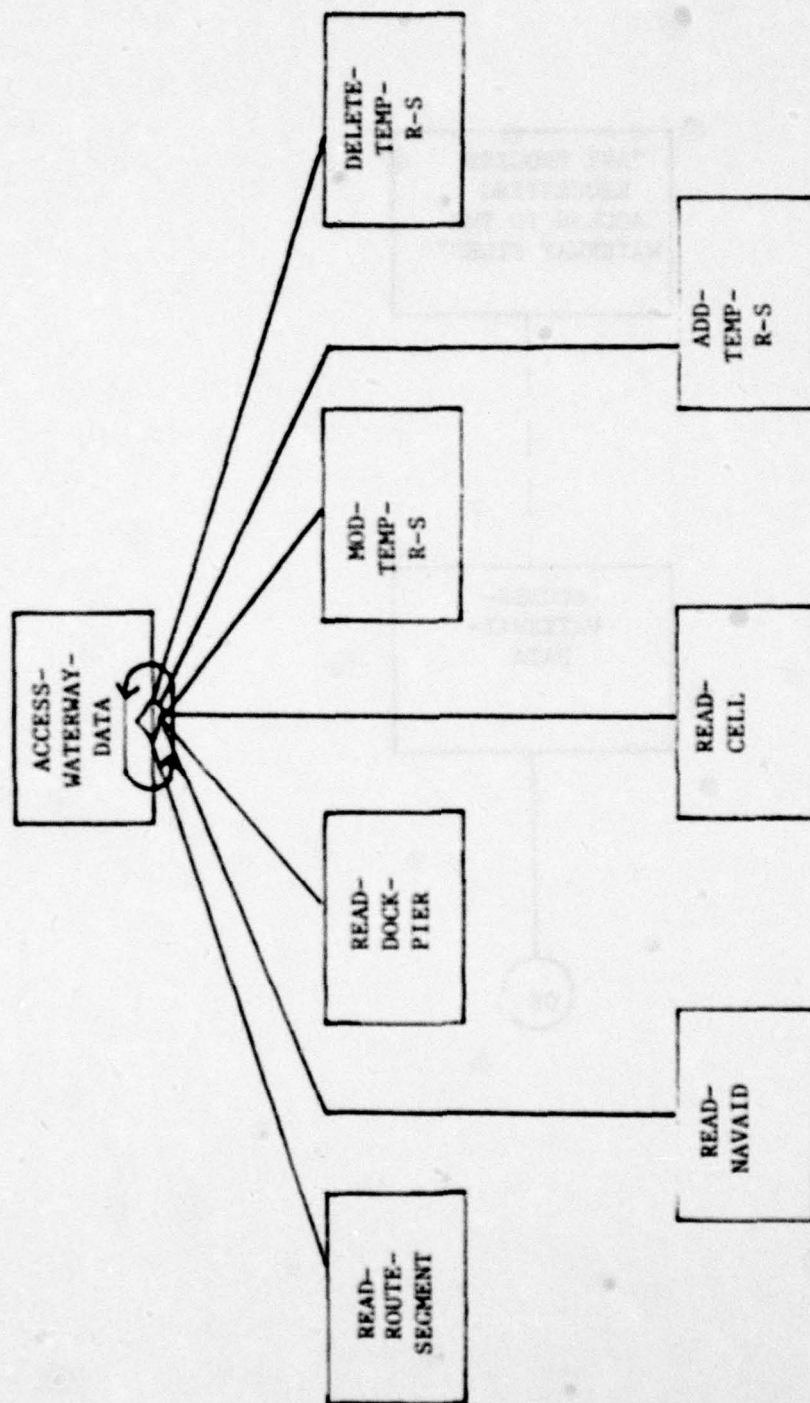


FIGURE 5-10B. ACCESS-WATERWAY-DATA



MESSAGE TYPE: Get-Dock-Pier-Record

FUNCTION: Read and return the contents of the specified dock/pier record.

INPUT: Designation of dock/pier.

ANSWER: Acknowledgement of message, flag indicating whether dock/pier record exists, and if found, the contents of the dock/pier record.

MESSAGE TYPE: Get-Navaid-Record

FUNCTION: Read and return the contents of the specified navaid record.

INPUT: Designation of navaid.

ANSWER: Acknowledgement of message, flag indicating whether navaid record exists, and if found, the contents of the record.

MESSAGE TYPE: Get-Cell-Data

FUNCTION: Read and return the data of the specified cell.

INPUT: Coordinates of cell.

ANSWER: Acknowledgement of message, flag indicating whether cell record is empty, and if not, the cell data.

MESSAGE TYPE: Enter-Temporary-Route-Segment

FUNCTION: Add a temporary route segment record to the file.

INPUT: Segment designation and other required data elements for the record.

ANSWER: Acknowledgement of message, and flag indicating whether route segment already exists.

MESSAGE TYPE: Modify-Temporary-Route-Segment  
FUNCTION: Modify data in a temporary route segment record.  
INPUT: Route Segment Designation and values of data elements to be changed.  
ANSWER: Acknowledgement of message and flag indicating whether record exists.

MESSAGE TYPE: Delete-Temporary-Route-Segment  
FUNCTION: Delete a temporary route segment record.  
INPUT: Route segment designation.  
ANSWER: Acknowledgement of message.



Process ACCESS-WATERWAY-DATA

begin

while true do

begin

if no message is queued then wait for a message;

case message type of

Get-Route-Segment-Record: READ-ROUTE-SEGMENT;

Get-Dock-Pier-Record: READ-DOCK-PIER;

Get-Navaid-Record: READ-NAVAID;

Get-Cell-Data: READ-CELL;

Enter-Temporary-Route-Segment: ADD-TEMP-R-S;

Modify-Temporary-Route-Segment: MOD-TEMP-R-S;

Delete-Temporary-Route-Segment: DELETE-TEMP-R-S;

end;

Send the answer to the message;

end;

end

Procedure READ-ROUTE-SEGMENT

Description: Get from the message buffer the route segment designation. Read the route segment record with the specified designation. If the record is found in the ROUTE-SEGMENT-FILE then copy the route segment record into the answer buffer, otherwise set the first word of the answer buffer to zero to indicate that the record does not exist.

Procedure READ-DOCK-PIER

Description: Get from the message buffer the dock/pier designation. Read the dock/pier record with the specified designation. If the record is found in the Dock-Pier-File then copy the dock/pier record into the answer buffer, otherwise set the first word of the answer buffer to zero to indicate that the record does not exist.

Procedure READ-NAVAID

Description: Get from the message buffer the navaid designation. Read the dock/pier record with the specified designation. If the record is found in the Navaid-File then copy the navaid record into the answer buffer, otherwise set the first word of the answer buffer to zero to indicate that the record does not exist.

Procedure READ-CELL

Description: Get from the message buffer the cell coordinates. Read the cell record. If the cell record is found in the Cell-File then copy the record into the answer buffer, otherwise set the first word of the answer buffer to zero to indicate that the record does not exist.



Procedure ADD-TEMP-R-S

Description: Get from the message buffer the temporary route segment designation. Check for a route segment with the same designation already in the Route-Segment-File. If a route segment with the same designation already exists then set a flag in the answer to indicate this; otherwise write the contents of the message buffer as a record in the Route-Segment-File, and clear the flag in the answer buffer.

Procedure MOD-TEMP-R-S

Description: Get the temporary route segment designation from the message buffer. Find the record in the Route-Segment-File with the same route segment designation. Write the contents of the message buffer into the route segment record. If the record was not found then set the flag in the answer buffer indicating this.

Procedure DELETE-TEMP-R-S

Description: Get the temporary route segment designation from the message buffer. Find the record in the Route-Segment-File and delete it.

#### 5.3.1.3 UPDATE-WATERWAY-FILES Process

This process is executed to change the "permanent" waterway data. The waterway data is stored in five data structures: 1) Route-Segment-File, 2) Waypoint-File, 3) Dock-Pier File, 4) Navaid-File, and 5) Cell-File. It is non-reentrant and is executed off-line while the system is inactive. While this process is running, all waterway files are locked out, preventing access by the ACCESS-WATERWAY-DATA process (described in Section 5.3.1.3). Input to the process is operator commands at the system console. The process consists of a set of procedures, each of which performs the function requested by the operator from the system console.

Whenever one of the functions is performed, the process must check for its effects on other records in the waterway file, e.g., if a waypoint is entered into the file and a route segment designation is specified in the in-bound route segment field, the waypoint record cannot be entered until the referenced route segment record is entered.

UPDATE-WATERWAY-FILES control/data paths are shown in Figure 5-11.



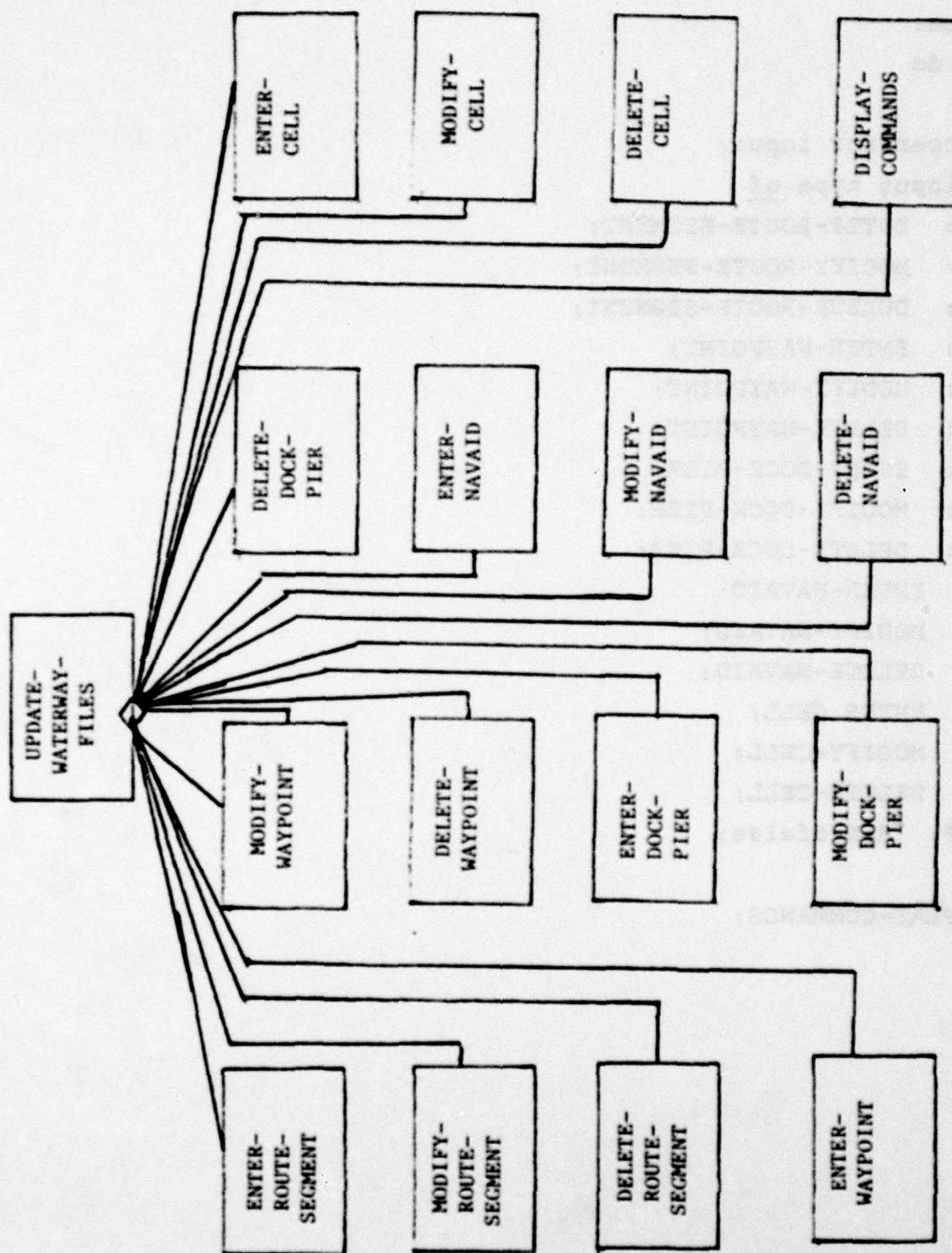


FIGURE 5-11. UPDATE-WATERWAY-FILES

Process UPDATE-WATERWAY-FILES

begin

RUN: = true;

while RUN do

begin

Read operator input;

Case input type of

ERS: ENTER-ROUTE-SEGMENT;

MRS: MODIFY-ROUTE-SEGMENT;

DRS: DELETE-ROUTE-SEGMENT;

EWP: ENTER-WAYPOINT;

MWP: MODIFY-WAYPOINT;

DWP: DELETE-WAYPOINT;

EDP: ENTER-DOCK-PIER;

MDP: MODIFY-DOCK-PIER;

DDP: DELETE-DOCK-PIER;

EN: ENTER-NAVAID;

MN: MODIFY-NAVAID;

DN: DELETE-NAVAID;

EC: ENTER CELL;

MC: MODIFY-CELL;

DC: DELETE-CELL;

STOP: RUN:=false;

else

DISPLAY-COMMANDS;

end

end

end



Procedure ENTER-ROUTE-SEGMENT

Description: Prompt for and input all data required to create a route segment record. Write the data as a record into the Route-Segment-File.

Procedure MODIFY-ROUTE-SEGMENT

Description: Prompt for and input the route segment designation. Read the route segment record. Display the contents of the record as a form, and allow the operator to alter fields in the record by typing over existing data. When the ENTER key is pressed, write the updated record over the old record in the Route-Segment-File.

Procedure DELETE-ROUTE-SEGMENT

Description: Prompt for and input the designation of the route segment to be deleted. Find and delete the specified record from the Route-Segment-File.

Procedure ENTER-WAYPOINT

Description: Prompt for and input all data required to create a waypoint record. Write the data as a record into the Waypoint-File.

Procedure    MODIFY-WAYPOINT

Description:    Prompt for and input the waypoint designation.  
Read the waypoint record.    Display the contents of the record  
as a form, and allow the operator to alter fields in the  
record by typing over existing data.    When the ENTER key is  
pressed, write the updated record over the old record in the  
Waypoint-File.

Procedure    DELETE-WAYPOINT

Description:    Prompt for and input the waypoint designation.  
Find and delete the Waypoint-Record.

Procedure    ENTER-DOCK-PIER

Description:    Prompt for and input all data required to create  
a dock/pier record.    Write the data as a record into the Dock-  
Pier-File.

Procedure    MODIFY-DOCK-PIER

Description:    Prompt for and input the dock/pier designation.  
Read the dock/pier record.    Display the contents of the record  
as a form, and allow the operator to alter fields in the record  
by typing over the existing data.    When the ENTER key is  
pressed, write the updated record over the old record in the  
file.



Procedure DELETE-DOCK-PIER

Description: Prompt for and input the dock/pier designation. Find and delete the dock/pier record from the Dock-Pier-File.

Procedure ENTER-NAVAID

Description: Prompt for and input all data required to create a navaid record. Write the data as a record into the Navaid-File.

Procedure MODIFY-NAVAID

Description: Prompt for and input the navaid designation. Find and read the navaid record. Display the contents of the record as a form and allow the operator to alter fields in the record by typing over the existing data. When the ENTER key is pressed, write the updated record over the old record in the file.

Procedure DELETE-NAVAID

Description: Prompt for and input the navaid designation. Find and delete the navaid record.

Procedure ENTER-CELL

Description: Prompt for and input the cell coordinates and all data to be placed in the cell record. Write the data as a record into the Cell-File.

Procedure MODIFY-CELL

Description: Prompt for and input the cell coordinates. Read the cell record. Display the contents of the record as a form, and allow the operator to alter fields in the record by typing over the existing data. When the ENTER key is pressed, write the updated record over the old record in the file.

Procedure DELETE-CELL

Description: Prompt for and input the cell coordinates. Find and delete the data from the cell record.

Procedure DISPLAY-COMMANDS

Description: Display on the terminal a list of the commands which may be entered, and the meaning of each command.



### 5.3.2 Environmental File Access

Included in this section are the data structures and the access process for the following environmental record types: 1) weather status, such as temperature and visibility; 2) waterway status, such as direction and speed of current and tide level; 3) weather forecast, and 4) environmental data manually entered. The information associated with each of these record types is a function of location within the waterway.

#### 5.3.2.1 Environmental Data Structures

Figure 5-12 shows the four record formats of the environmental file, which are maintained in four sequential files, i.e., the Weather-Station-File, the Current-Tide-File, the Forecast-File and the Manual-Environment-File.

<u>DATA ELEMENT</u>	<u>UNITS</u>	<u>LENGTH (BYTES)</u>
<u>Automatic Weather Stations</u>		
Designation	Characters	14
Location	Coordinates	4
Temperature	Degrees	1
Visibility	Miles	1
Precipitation Rate	Scaled	2
Wind Direction	Degrees	2
Wind Speed		1
TOTAL		<u>25</u>
<u>Automatic Current and Tide Sensors</u>		
Designation	Characters	14
Location	Coordinates	4
Direction of Current	Degrees	2
Speed of Current		1
Tide Level Referred to MLLW		1
Average Wave Height		
TOTAL		<u>22</u>
<u>Manual Data Inputs</u>		
Source of Data	Characters	36
Date/Time Entered		4
Location Where Valid	Sector(s)	1
	Coordinates	4
Key Words	Characters	14
Free Form Text	Characters	160
TOTAL		<u>219</u>
<u>Forecasts</u>		
Source	Characters	36
Date/Time Entered		4
Date/Time Span Valid		8
Area Covered	Sector(s)	2
Forecast	Characters	160
TOTAL		<u>210</u>

Figure 5-12. Environmental File Records



### 5.3.2.2 ACCESS-ENVIRONMENTAL-DATA Process

All access to the environmental file is performed by the ACCESS-ENVIRONMENTAL-DATA process. A process desiring access sends a message to this process with the specified function and necessary information. The ACCESS-ENVIRONMENTAL-DATA process is reentrant; thus, it may be processing several such message requests concurrently.

ACCESS-ENVIRONMENTAL-DATA control/data paths are shown in Figure 5-13 A/B.

#### Message Types:

The process consists of a set of procedures, each of which performs the function requested by one of the following message types:

MESSAGE TYPE: Get-Automatic-Weather-Record

FUNCTION: Read and return the contents of the specified automatic weather record.

INPUT: Weather station designation.

ANSWER: Acknowledgement of the message, a flag indicating whether a record exists for such a weather station designation and if so, the contents of the record.

MESSAGE TYPE: Enter-Automatic-Weather Record

FUNCTION: Add an automatic weather record to the file.

INPUT: Weather station designation and location (coordinates).

ANSWER: Acknowledgement of the message and a flag indicating whether the record already exists.

MESSAGE TYPE: Modify-Automatic-Weather-Record

FUNCTION: Change weather data in a weather record.

INPUT: Weather station designation and data elements.

ANSWER: Acknowledgement of the message, and a flag indicating whether the record exists.

"ANY PROCESS  
REQUESTING  
ACCESS TO THE  
ENVIRONMENTAL  
DATA FILES"

ACCESS-  
ENVIRONMENTAL-  
DATA

OS

Figure 5-13A. ACCESS-ENVIRONMENTAL-DATA



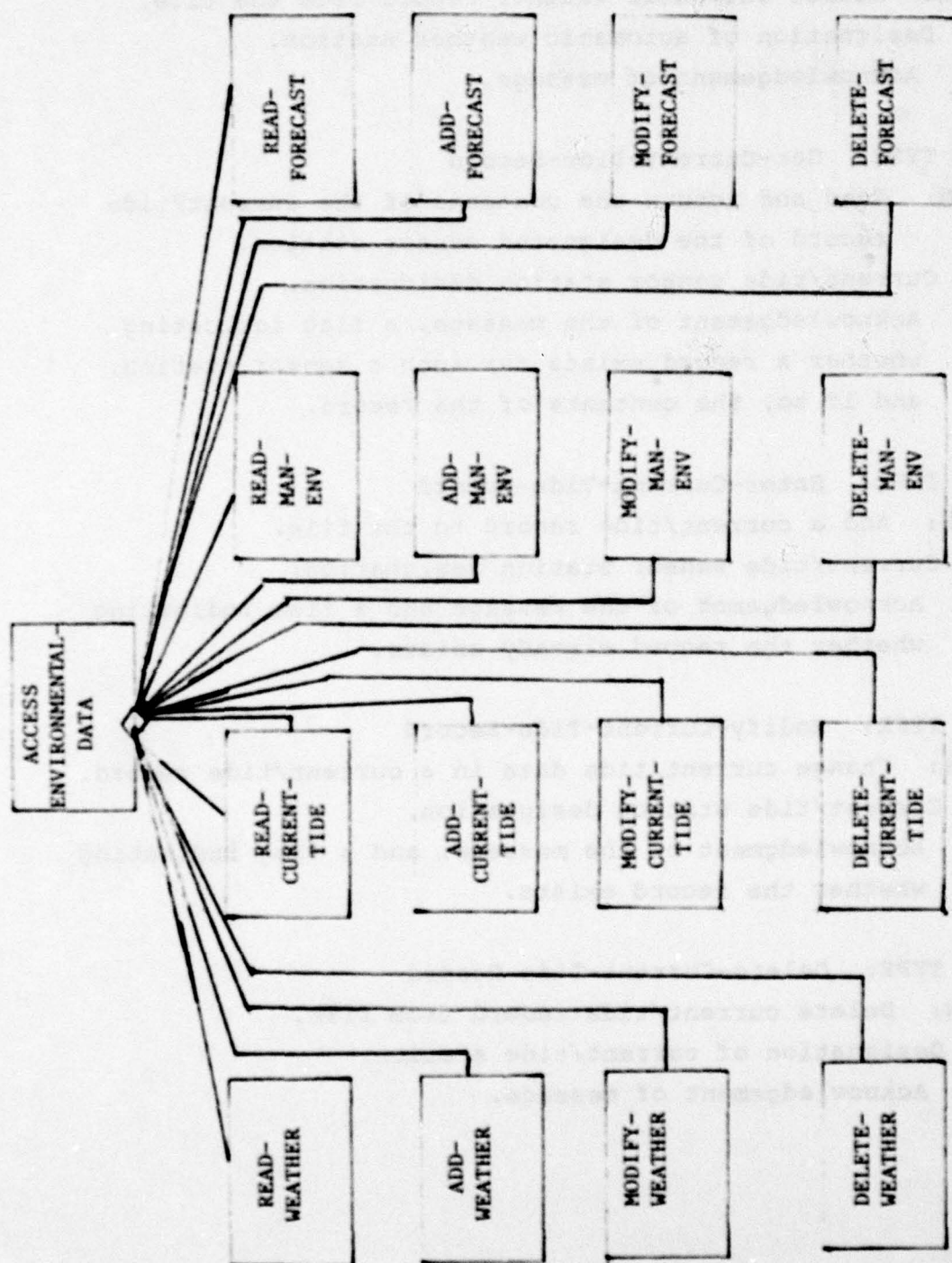


Figure 5-13B. ACCESS-ENVIRONMENTAL-DATA

MESSAGE TYPE: Delete-Automatic-Weather-Record  
FUNCTION: Delete automatic weather record from the file.  
INPUT: Designation of automatic weather station.  
ANSWER: Acknowledgement of message.

MESSAGE TYPE: Get-Current-Tide-Record  
FUNCTION: Read and return the contents of the current/tide  
record of the designated sensor station.  
INPUT: Current/tide sensor station designation.  
ANSWER: Acknowledgement of the message, a flag indicating  
whether a record exists for such a sensor station,  
and if so, the contents of the record.

MESSAGE TYPE: Enter-Current-Tide-Record  
FUNCTION: Add a current/tide record to the file.  
INPUT: Current/tide sensor station designation.  
ANSWER: Acknowledgemnt of the message and a flag indicating  
whether the record already exists.

MESSAGE TYPE: Modify-Current-Tide-Record  
FUNCTION: Change current/tide data in a current/tide record.  
INPUT: Current/tide station designation.  
ANSWER: Acknowledgment of the message, and a flag indicating  
whether the record exists.

MESSAGE TYPE: Delete-Current-Tide Record  
FUNCTION: Delete current/tide record from file.  
INPUT: Designation of current/tide station.  
ANSWER: Acknowledgement of message.



MESSAGE TYPE: Get-Manual-Environment-Records

FUNCTION: Read and return the contents and record number of all the manually entered environment record specified by the sector and the time/date range(optional),

INPUT: Sector designation and time/date range (optional)

ANSWER: Acknowledgement of a message, count of number of records found meeting the specification, and the list of records found with their relative record numbers in the file.

MESSAGE TYPE: Enter-Manual-Environment-Record

FUNCTION: Add a manual environment record to the file.

INPUT: Contents of the record.

ANSWER: Acknowledge of message.

MESSAGE TYPE: Modify-Manual-Environment-Record

FUNCTION: Change specified data elements in the record specified by the relative record number in the file.

INPUT: Relative record number in file, and values of data elements to be changed.

ANSWER: Acknowledgement of message, and a flag indicating whether record exists.

MESSAGE TYPE: Delete-Manual-Environment-Record

FUNCTION: Delete a manual environment record from the file.

INPUT: Relative record number.

ANSWER: Acknowledgement of record.

MESSAGE TYPE: Get-Forecast-Record

FUNCTION: Read and return the contents and record number of the forecast records specified by the sector number and the time/date span.

INPUT: Sector number and time/date span.

ANSWER: Acknowledgement of the message, count of the number of qualifying records found, and list of the contents and relative record number of each such record.

MESSAGE TYPE: Enter-Forecast-Record

FUNCTION: Add a forecast record to the file.

INPUT: Contents of record to be entered.

ANSWER: Acknowledgement of the message.

MESSAGE TYPE: Modify-Forecast-Record

FUNCTION: Modify a forecast record.

INPUT: Relative record number, and value of data elements to be changed.

ANSWER: Acknowledgement of message.

MESSAGE TYPE: Delete-Forecast-Record

FUNCTION: Delete a forecast record from the file.

INPUT: Relative record number of records to be deleted.

ANSWER: Acknowledgement of message.



Process ACCESS-ENVIRONMENTAL-DATA

begin

while true do

begin

if no message is queued then wait for a message;

case message type of

Get-Automatic-Weather-Record: READ-WEATHER;

Enter-Automatic-Weather-Record: ADD-WEATHER;

Modify-Automatic-Weather-Record: MODIFY-WEATHER;

Delete-Automatic-Weather-Record: DELETE-WEATHER;

Get-Current-Tide-Record: READ-CURRENT-TIDE;

Enter-Current-Tide-Record: ADD-CURRENT-TIDE;

Modify-Current-Tide-Record: MODIFY-CURRENT-TIDE;

Delete-Current-Tide-Record: DELETE-CURRENT-TIDE;

Get-Manual-Environment-Record: READ-MAN-ENV;

Enter-Manual-Environment-Record: ADD-MAN-ENV;

Modify-Manual-Environment-Record: MODIFY-MAN-ENV;

Delete-Manual-Environment-Record: DELETE-MAN-ENV;

Get-Forecast-Record: READ-FORECAST;

Enter-Forecast-Record: ADD-FORECAST;

Modify-Forecast-Record: MODIFY-FORECAST;

Delete-Forecast-Record: DELETE-FORECAST;

end;

Send the answer;

end;

end

Procedure READ-WEATHER

Description: Get from the message buffer the weather station designation. Read the weather station record containing the specified designation. If the record is found in the Weather-Station-File, copy the record into the answer buffer; otherwise, set the flag in the answer buffer to indicate that the record was not found.

Procedure ADD-WEATHER

Description: Get from the message buffer the weather station designation and location. If a weather station record with the same designation or location exists in the Weather-Station-File, set the flag in the answer buffer to indicate this; otherwise, write the contents of the message buffer as a record in the Weather-Station-File, and clear the flag in the answer buffer.

Procedure MODIFY-WEATHER

Description: Get the weather station designation from the message buffer. Find the record in the Weather-Station-File with the same designation. Write the contents of the message buffer into the weather station record. If the record was not found, set the flag in the answer buffer indicating this condition.



Procedure DELETE-WEATHER

Description: Get the weather station designation from the message buffer. Find the record in the Weather-Station-File and delete it.

Procedure READ-CURRENT-TIDE

Description: Get from the message buffer the current/tide sensor station designation. Read the current/tide station record containing the specified designation. If the record is found in the Current-Tide-File, copy the record into the answer buffer; otherwise, set the flag in the answer buffer to answer buffer to indicate that the record was not found.

Procedure ADD-CURRENT-TIDE

Description: Get the current/tide sensor station designation from the message buffer. Search the Current-Tide-File for a record with the same designation. If such a record is found, set the flag in the answer buffer to indicate this condition; otherwise, write the contents of the message buffer as a record in the Route-Segment-File and clear the flag in the answer buffer.

Procedure    MODIFY-CURRENT-TIDE

Description:    Get the current/tide station designation from the message buffer. Find the record in the Current-Tide-File with the same designation. Write the contents of the message buffer into the current/tide record. If the record was not found, set the flag in the answer buffer indicating this condition.

Procedure    DELETE-CURRENT-TIDE

Description:    Get the current/tide station designation from the message buffer. Find the record in the Current-Tide-File and delete it.

Procedure    READ-MAN-ENV

Description:    Get the sector number and time/date span from the message buffer. Search the Manual-Environment-File for all records with the same sector number and a time/date within the specified time/date span. For each qualifying record found, copy the record and the record number into the answer buffer, and increment in the answer buffer a count of the number of records found.

Procedure    ADD-MAN-ENV

Description:    Write the contents of the message buffer as a record in the Manual-Environment-File.



Procedure    MODIFY-MAN-ENV

Description:    Get the relative record number from the message buffer.    Read the record in the Manual-Environment-File specified by the relative record number.    If the record is empty (contains no data), set a flag in the answer buffer indicating this condition; otherwise, substitute the values specified in the message buffer into the record buffer, and write the updated record into the Manual-Environment File.

Procedure    DELETE-MAN-ENV

Description:    Get the relative record number from the message buffer.    Write a zero-filled record into the Manual-Environment-File at the file location specified by the record number.

Procedure    READ-FORECAST

Description:    Get the sector number and time/date span from the message buffer.    Search the Forecast-File for all records with the same sector number and a time/date within the specified time/date span.    For each qualifying record found, copy the record and the record number into the answer buffer, and increment in the answer buffer a count of the number of records found.

Procedure    ADD-FORECAST

Description:    Write the contents of the message buffer as a record in the Manual-Environment-File.

Procedure MODIFY-FORECAST

Description: Get the relative record number from the message buffer. Read the record in the Forecast-File specified by the record number. If the record is empty (contains no data), set a flag in the answer buffer indicating this condition; otherwise, substitute the values specified in the message buffer into the record buffer, and write the updated record into the Forecast-File.

Procedure DELETE-FORECAST

Description: Get the relative record number from the message buffer. Write a zero-filled record into the Forecast-File at the location specified by the record number.



### 5.3.3 Notices File Access

The data structures and access process of the notices file are described in this section.

#### 5.3.3.1 Data Structures

The notices file may be accessed by sector, notice type, and time/date span. A matrix indexed by sector and type is core-resident. Each element of the matrix is either empty or contains a pointer to a list of disc addresses of those records of the specified type which apply to the specified sector. The list is in order of time/date entered. The content and format of a notices file record is as follows:

<u>DATA ELEMENT</u>	<u>UNITS</u>	<u>LENGTH (BYTES)</u>
Type of Notice	Types	1
Light List Number if Applicable	Characters	36
Portion(s) Covered by Notice	Sectors	2
Name or Number of Navaid if Applicable	Characters	36
Date/Time Entered		4
Date/Time Span Valid		8
Text	Characters	10,000
TOTAL		10,087

#### 5.3.3.2 ACCESS-NOTICES-FILE Process

This process accesses the notices records using the in-core structure described in 5.3.3.1. It is reentrant, allowing multiple concurrent accesses.

ACCESS-NOTICES-FILE control/data paths are shown in Figure 5-14 A/B.

"ANY PROCESS  
REQUESTING  
ACCESS TO THE  
NOTICES FILE"

ACCESS-  
NOTICES-  
FILE

OS

FIGURE 5-14A. ACCESS-NOTICES-FILE



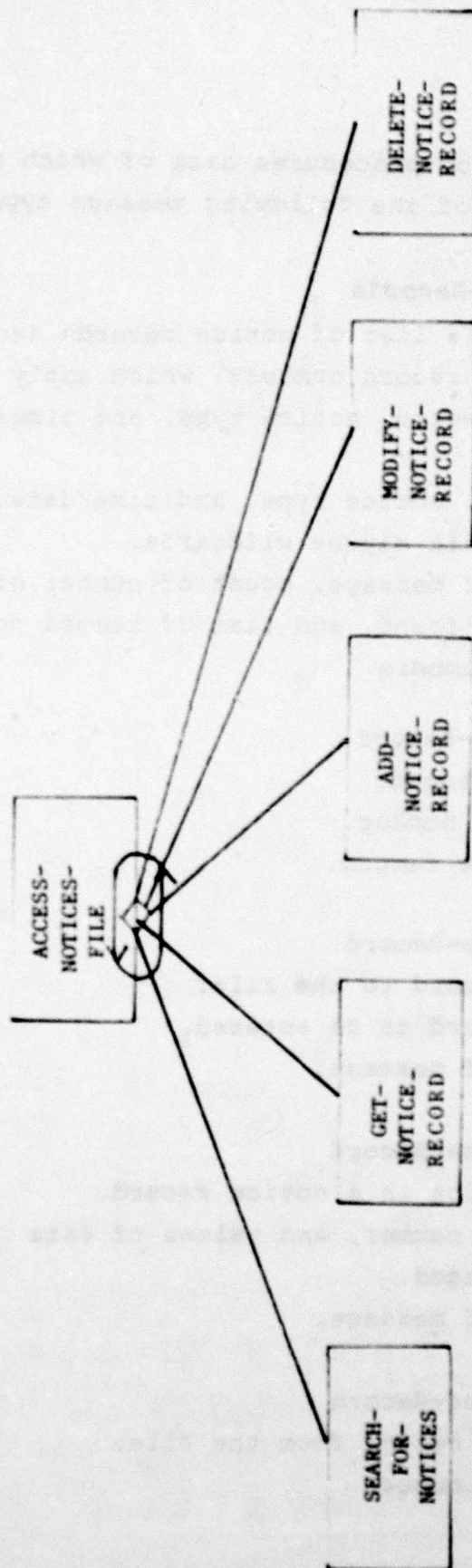


FIGURE 5-14B. ACCESS-NOTICES-FILE

Message Types:

The process consists of a set of procedures each of which performs the function requested by one of the following message types:

MESSAGE TYPE: Get-Notice-Records

FUNCTION: Read and return a list of notice records (and their disc file record numbers) which apply to the specified sector, notice type, and time/date span.

INPUT: Sector designation, notice type, and time/date.  
Any of these criteria may be wildcards.

ANSWER: Acknowledgement of message, count of number of qualified records found, and list of record contents and file record numbers.

MESSAGE TYPE: Read-Notice-Record

FUNCTION: Read a notice record.

INPUT: Notice file record number.

ANSWER: Contents of notice record.

MESSAGE TYPE: Enter-Notice-Record

FUNCTION: Add a notice record to the file.

INPUT: Copy of notice record to be entered.

ANSWER: Acknowledgement of message.

MESSAGE TYPE: Modify-Notice-Record

FUNCTION: Change information in a notice record.

INPUT: Notice file record number, and values of data elements to be changed.

ANSWER: Acknowledgement of message.

MESSAGE TYPE: Delete-Notice-Record

FUNCTION: Remove a notice record from the file.

INPUT: Notice file record number.

ANSWER: Acknowledgement.



Process ACCESS-NOTICES-FILE

```
begin  
  while true do  
    begin  
      if no messages queued then wait for a message;  
      case message type of  
        Get-Notice -Records: SEARCH-FOR-NOTICES;  
        Read-Notice-Record: GET-NOTICE-RECORD;  
        Enter-Notice-Record: ADD-NOTICE-RECORD;  
        Modify-Notice-Record: MODIFY-NOTICE-RECORD;  
        Delete-Notice-Record: DELETE-NOTICE-RECORD;  
      end;  
      Send the answer;  
    end;  
  end
```

Procedure SEARCH-FOR-NOTICES

Description: Get the sector number, notice type, and time/date span from the message buffer. Search the Notices-File for all records with the specified sector and type, and a time/date within the specified span. For each qualifying record found, copy the record and the record number into the answer buffer, and increment in the answer buffer a count of the number of records found.

Procedure GET-NOTICE-RECORD

Description: Get the record number from the message buffer, and read the record specified by the record number. Place the contents of the record in answer buffer.

Procedure ADD-NOTICE-RECORD

Description: Write the contents of the message buffer as a record in the Notices-File.

Procedure MODIFY-NOTICE-RECORD

Description: Get the record number from the message buffer. Read the record in the Notices-File specified by the record number. If the record is empty (contains no data), set a flag in the answer buffer indicating this condition; otherwise, substitute the values specified in the message buffer into the record buffer, and write the updated record into the Notices-File.



Procedure DELETE-NOTICE-RECORD

Description: Get the record number from the message buffer.  
Write a zero-filled record into the Notices-File at the file  
location specified by the record number.

#### 5.3.4 Display Station Status Access

The ACCESS-DISPLAY-STATION-STATUS process is invoked to access the Display-Station-Status-Table (D-S-S-T) in the main processor.

ACCESS-DISPLAY-STATION-STATUS control/data paths are shown in Figure 5-15A/B.

#### Data Structures:

The D-S-S-T contains an entry for each display station. Each entry consists of the following data elements:

- 1) Display Station ID.
- 2) Watchstander ID (0 if not logged on).
- 3) Supervisor Flag (set if watchstander is the watch supervisor).
- 4) Designation of Sector (as assigned to watchstander).

#### Message Types:

The process is comprised of a set of procedures, each of which performs the function requested by one of the following message types:

MESSAGE TYPE: Log-On

FUNCTION: Enters data in a D-S-S-T entry

INPUT: 1) Display Station ID

2) Watchstander ID

3) Supervisor Flag

4) Sector Designation

ANSWER: Acknowledgement of message and a flag indicating whether a watchstander is already logged onto the display station.



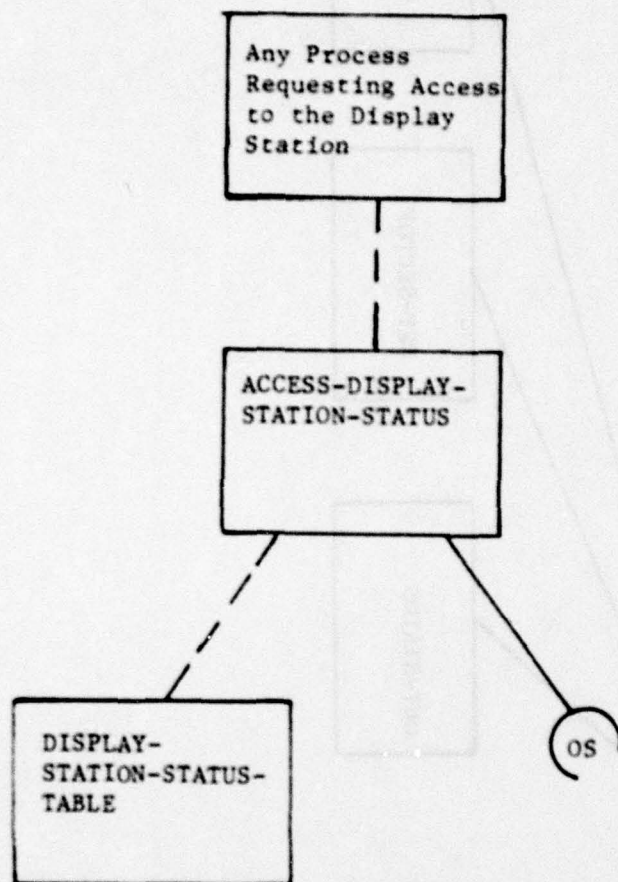


Figure 5-15A. ACCESS-DISPLAY-STATION-STATUS

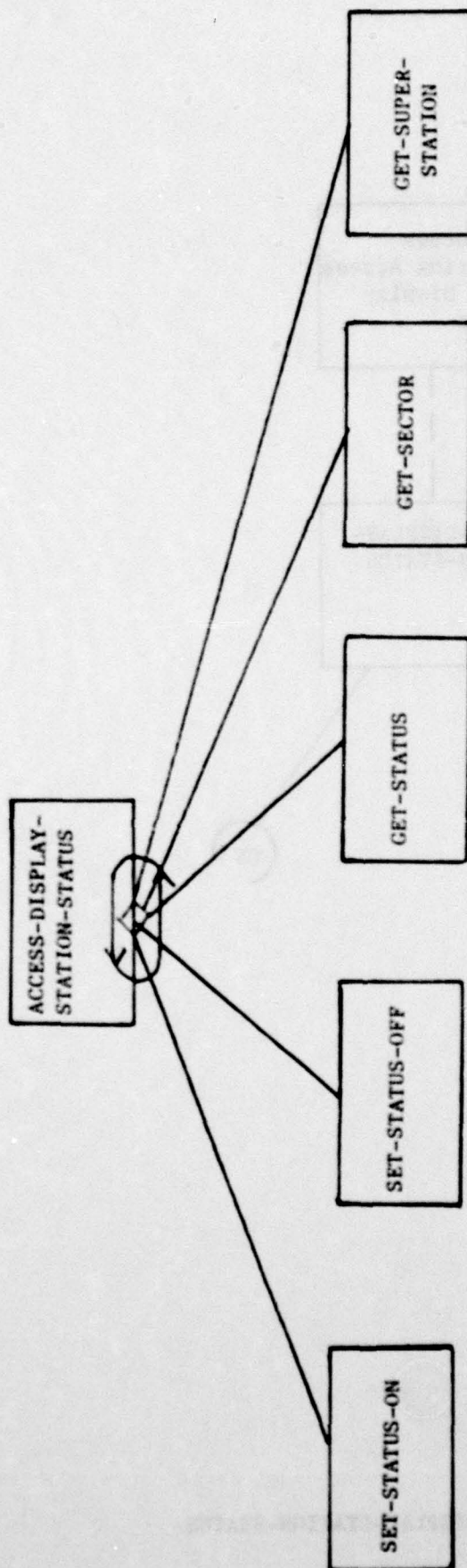


Figure 5-15B. ACCESS-DISPLAY-STATION-STATUS



MESSAGE TYPE: Log-Off

FUNCTION: Remove (zero-fill) watchstander, supervisor flag,  
and sector fields in D-S-S-T entry.

INPUT: Display station ID.

ANSWER: Acknowledgement of message.

MESSAGE TYPE: Get-Display-Station-Status

FUNCTION: Return status information of display station.

INPUT: Display station ID.

ANSWER: Acknowledgement of message and content of D-S-S-T  
entry.

MESSAGE TYPE: Get-Sector-Assignment

FUNCTION: Determine which display station is assigned to  
the specified sector.

INPUT: Sector designation.

ANSWER: Acknowledgement of message, flag indicating whether  
a display station is assigned to the sector, and if  
so, then the content of the D-S-S-T entry.

MESSAGE TYPE: Get-Supervisor-Station

FUNCTION: Determine onto which display station the watch  
supervisor is logged.

INPUT: None.

ANSWER: Acknowledgement of message, and D-S-S-T entry of  
supervisor.

Process ACCESS-DISPLAY-STATION-STATUS

begin

while true do

begin

if no message is queued then wait for a message;

case message type of

Log-On: SET-STATUS-ON;

Log-Off: SET-STATUS-OFF;

Get-Display-Station-Status: GET-STATUS;

Get-Sector-Assignment: GET-SECTOR;

Get-Supervisor-Station: GET-SUPER-STATION;

end;

Send the answer;

end;

end



AD-A078 937

INTERNATIONAL COMPUTING CO BETHESDA MD F/G 15/5  
VESSEL TRAFFIC SERVICES PROCESSING/DISPLAY SUBSYSTEM SOFTWARE R--ETC(U)  
SEP 79 C C HENSON , R S GRAHAM , B A MCINTOSH DOT-CG-81-78-1833

USCG -D-73-79

NL

UNCLASSIFIED

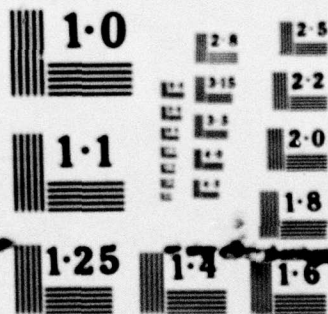
5 OF 5  
AD-A078937



END  
DATE  
FILMED

1-80

DDC



NATIONAL BUREAU OF STANDARDS  
MICROCOPY RESOLUTION TEST CHART



Procedure SET-STATUS-ON

Description: Get the display station ID from the message buffer. Find the entry in the D-S-S-T with the same display station ID. If the watchstander ID in the D-S-S-T entry is zero (not logged on) then move the watchstander ID, the supervisor flag, and the sector designation from the message buffer to the D-S-S-T entry. Otherwise if the watchstander ID in the D-S-S-T entry is not zero (already logged on) then set a flag in the answer buffer indicating so.

Procedure SET-STATUS-OFF

Description: Get the display station ID from the message buffer. Find the entry in the D-S-S-T with the same display station ID. Zero-fill all fields in the D-S-S-T entry except the display station ID.

Procedure GET-STATUS

Description: Get the display station ID from the message buffer. Find the entry in the D-S-S-T with the same display station ID. Move the contents of the D-S-S-T entry to the answer buffer.

Procedure GET-SECTOR

Description: Get the sector designation from the message buffer. Find the entry in the D-S-S-T with the same sector designation. If an entry is found which is assigned to the sector then move the D-S-S-T entry contents to the answer buffer, otherwise set a flag in the answer buffer indicating that no display station is currently assigned to the sector.

Procedure GET-SUPER-STATION

Description: Find the entry in the D-S-S-T with the supervisor flag set. Move the contents of the D-S-S-T entry to the answer buffer.



### 5.3.5 Key Search

The SEARCH-ON-KEY process performs a search on request from a TRANSACT-WATCHSTANDER-REQUESTS (T-W-R) process. It resides as a low priority process in the main processor. The following describes the message it receives from T-W-R.

MESSAGE TYPE: Search-For-Key-Values

FUNCTION: Perform a file search based on the given parameters.

INPUT: 1) File Name.

2) List of key-value relationships.

3) List of data elements (record fields) to be returned/output.

4) Line printer flag.

ANSWER: Immediate acknowledgement of message.

Upon receiving such a message the SEARCH-ON-KEY process validates the message contents, and acknowledges the message. It then searches the specified file sequentially. For each record whose values match the specified values for the specified fields, the process extracts the data from the specified output fields. With these values it builds an output record and writes it to a temporary file. When the search is complete, then the records in the temporary file are sorted lexicographically with respect to the output fields.

If the number of output records is greater than a preset threshold, or if the line printer flag is set, then the output file is sent to the line printer handler for printing. A message is sent to the MANAGE-ACTION-REQUIRED-LIST process to Add-Entry-to-List of the display station which initiated the request. The entry will be a notification that the search is complete, and if the output was not directed to the line printer, then it will be contained within the message to the M-A-R-L process. The watchstander may then review the output on his alphanumeric display at his request.

SEARCH-ON-KEY control/data paths are shown in Figure 5-16.



TRANSACT-  
WATCHSTANDER-  
REQUEST

SEARCH-ON-KEY

OS

Figure 5-16. SEARCH-ON-KEY

Process SEARCH-ON-KEY

begin

while true do

begin

if no message queued then wait on a message;  
validate the message contents;  
send an answer acknowledging receipt of the message;  
Create a temporary output file in which to place  
the search output data;  
Open the file to be searched for read-only access;  
Output-Count: = 0;  
End-of-File: = false;  
Read the first record in the file being searched;  
while not End-of-File do

begin

Output-flag: = true  
repeat until all search keys have been tested  
or Output-Flag = false do  
if data field in record does not satisfy  
the corresponding search parameter  
then output-Flag: = false;  
if Output-Flag = true.  
then

begin

for each record field to be output do  
Format and place the field in the  
Output-Buffer;  
Write the Output-Buffer to the temporary file'  
Increment the Output-Count;

end;



```

    Read the next record in the file;
    if end of file is encountered
    then End-of-File: = true;

    end;

    Sort the temporary file records lexicographically;
    Get the Line-Printer-Flag from the message buffer;
    Clear the message buffer;

    if Output-Count = 0
    then Place the message 'NO SEARCH RECORDS FOUND' in message
        buffer;
    if Output-Count > threshold for display station output
    then Line-Printer-Flag: = true;
    if Line-Printer-Flag = true
    then
        begin
            Request the operating system to output the contents of
            the temporary file to the line printer;
            Place the message 'SEARCH OUTPUT ON LINE PRINTER' in
            the message buffer;
        end
    else
        Place the message 'SEARCH RESULTS' and the contents of the
        temporary file in the message buffer;
        Send the contents of the message buffer as an Add-Entry-to-
        List message to the MANAGE-ACTION-REQUIRED-LIST process;
    end;
end

```

## 5.4 DEMAND FUNCTIONS

In addition to the functions described in Section 5 which enter and retrieve information from the system, other functions will be provided which perform calculations, analysis and other processing based on information stored in the VTS Processing/Display Subsystem.

If the information is readily available in the display station processor, the TRANSACT-WATCHSTANDER-REQUEST process (Section 4.1) will perform the function itself. CPA calculations and relative position determinations, for example, can normally be done by the TRANSACT-WATCHSTANDER-REQUEST process. Other functions will be performed by individual processes in the main processor.

### 5.4.1 DETERMINE-ENCOUNTERS

The DETERMINE-ENCOUNTERS process receives a message from the TRANSACT-WATCHSTANDER-REQUEST process (Section 4.1) which includes a vessel and a look-ahead time span. The location, course and speed of other vessels within the VTS coverage area will be compared against that of the specified vessel. An answer will then be formulated which includes all vessels which the specified vessel will meet, cross or overtake during the look-ahead time.

DETERMINE-ENCOUNTERS control/data paths are shown in Figure 5-17.



TRANSACT-  
WATCHSTANDER-  
REQUEST

DETERMINE-  
ENCOUNTERS

OS

Figure 5-17. DETERMINE-ENCOUNTERS

Process      DETERMINE-ENCOUNTERS

begin

while true do

begin

if no message is queued then wait for a message;

      Get the vessel internal ID and look ahead time span from the message;

      Max-Dist: = a weighted function of the look-ahead time span;

for each vessel in the Position-Table do

begin

          dx: = difference in latitude between the subject vessel  
              (vessel for which we are determining the encounters)  
              and the entry in the Position-Table;

          dy: = difference in longitude between the subject  
              vessel and the entry in the Position-Table;

if dx < Max-Dist and dy < Max-Dist

then add the internal ID to the possible encounters list;

end

        Create a list of route segments which lie within Max-Dist  
        of the current position of the subject vessel;

for each vessel in the possible encounters list do

begin

            Get the vessel intended route segments;

if there are no common route segments between the  
              vessel's intended route and the created list of  
              route segments

then remove the entry from the possible encounters list

end;

for each vessel remaining in the possible encounters list do

begin

            Calculate the location and time of the encounter based  
            on intended route;

if there will be an encounter

then

begin



Determine whether the situation will be an overtaking,  
a crossing, or a meeting;  
Determine whether the lane at the encounters point  
is 1 way or 2 way;  
Get the vessel name, type, cargo, and length;  
Add an entry to the Predicted-Encounters-List  
consisting of the encounter time, encounter location,  
vessel name, vessel type, vessel cargo, situation  
type, and 1 or 2 way lane;  
end;  
end;  
Sort the Predicted Encounters-List chronologically;  
Move the Predicted-Encounters-List to the answer buffer;  
Send the answer;  
end;  
end

#### 5.4.2 ACCESS-SYSTEM-PARAMETERS

The ACCESS-SYSTEM-PARAMETERS process will receive messages from the TRANSACT-WATCHSTANDER-REQUEST process which is interacting with the Watch Supervisor. The message may include a designation for the parameter(s) to be changed and the new value(s) to be assigned. Figure 5-18 shows a list of the parameters which are used to define and adjust the operation of the system. Included are parameters which define the rate at which hazard detection processes operate and parameters which define constricted areas.

To allow maximum flexibility an extensive set of operations will be available to the Watch Supervisor. Since a considerable amount of software may be needed for this process and since only one individual at a time may be using these functions, it may be appropriate to overlay the programs which support this process.

When the function has been completed, an acknowledgement message (answer) will be returned to the TRANSACT-WATCHSTANDER-REQUEST process.

ACCESS-SYSTEM-PARAMETERS control/data paths are shown in Figure 5-19.



- 1) Cycle Times for Hazard Process
- 2) Stage-1 Collision Distance Threshold
- 3) Stage-2 Collision CPA Threshold
- 4) Stage-2 Collision tCPA Threshold
- 5) Vessel Size Risk Values
- 6) Cargo Hazard Parameters
- 7) Risk Weights
- 8) Lane Stray Look-Ahead Time
- 9) Route Stray Threshold Distance
- 10) Grounding Threshold
- 11) Grounding Look-Ahead Time
- 12) Excessive Congestion Look-Ahead Time
- 13) Excessive Congestion Capacity Contribution Factors  
(for each vessel size)
- 14) Dangerous Encounters Look Ahead Time
- 15) Channel Crossing Threshold Time
- 16) Dangerous Encounter Size Combination Threshold
- 17) Navaid List for Adrift/Missing Processing
- 18) Speeding Threshold
- 19) Sensor Logging Cycle Time
- 20) Local Traffic Default Radius
- 21) Local Traffic Maximum Radius
- 22) CPA Time Threshold
- 23) Map Update Cycle Time
- 24) Alert Timeout Priority
- 25) Alert Response Timeout

Figure 5-18. System Parameter List

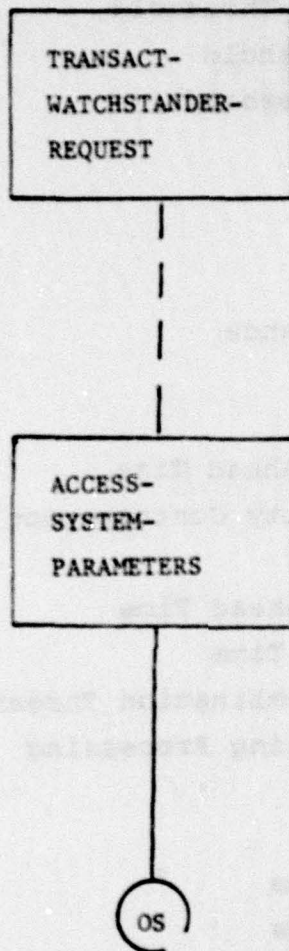


Figure 5-19. ACCESS -SYSTEM-PARAMETERS



Process ACCESS-SYSTEM-PARAMETERS

```
begin
  while true do
    begin
      if no message is queued then wait on a message;
      Get type of system parameter from the message buffer
      (see Figure 5-18 for list of possible system parameters);
      if the message specifies a write operation
      then
        Write the system parameter value contained in the message
        into the proper record in the System-Parameters-File
      else
        Read the system parameter value from the proper record
        in the System-Parameter-File, and place the value in the
        answer buffer;
        Send the answer;
      end
    end
  end
```

## 5.5 SIMULATION

There are two processes which perform the simulation functions requested through the TRANSACT-WATCHSTANDER-REQUESTS process. The SET-UP-SCENARIO process performs the creation and modification of a scenario record. The PLAYBACK-SCENARIO process controls the playback of a scenario recorded by the SET-UP-SCENARIO process.

### 5.5.1 Data Structures

The major data structure is the scenario file, which consists of a set of scenario records, one for each scenario.

A scenario record consists of the following information:

- 1) Scenario name - maximum 25 characters.
- 2) Artificial vessel list - maximum 100 vessels.
  - 2.1) Vessel name.
  - 2.2) Vessel movement list - maximum 25 entries.
    - 2.2.1) Route.
      - 2.2.1.1.) Route segment designation or position coordinates.
      - 2.2.1.2) Elapsed time from start of scenario when vessel is to be at start of route segment or coordinates.
      - 2.2.1.3) Rate of change of bearing.
      - 2.2.1.4) Rate of change of speed.
- 3) Live vessel list - maximum 100 vessels
  - 3.1) Vessel Name (blank if unidentified)
  - 3.2) Start time - elapsed time from start of scenario when recording of vessel is started.
  - 3.3) End time - elapsed time from start of scenario when recording of vessel is stopped.
- 4) Simulated vessel file designation.
- 5) Simulated passage file designation.
- 6) Weather data records
  - maximum of 19 records
  - each record contains 10 minutes of average of recorded data.



7) Current and Tide Data

- maximum of 19 records
- each record contains 10 minute average of recorded data.

5.5.2 SET-UP-SCENARIO Process

This process is used to create and modify a scenario record. It may receive any of the following message types from the TRANSACT-WATCHSTANDER-REQUESTS process. Upon receiving such a message, it checks with the DISPLAY-STATION-STATUS-TABLE process to verify that the Watch Supervisor is logged onto the display station requesting the function.

MESSAGE TYPE: Record-Scenario

FUNCTION: The process records live vessel movement data and sensor data. It uses this data, along with artificial vessel data to create a scenario record.

INPUT: All scenario record information, except that which is to be recorded. Time period for which data is to be recorded.

ANSWER: Acknowledges message. When recording is complete, the process sends a message to the ACTION-REQUIRED process to notify the watchsupervisor of the completion.

MESSAGE TYPE: Modify-Scenario

FUNCTION: The process makes changes to an existing scenario record. If any live vessels are added to the scenario, then the live vessel's movements are recorded.

INPUT: Name of scenario record and changes to be made, i.e., adding, deleting or changing data on live or artificial vessels.

ANSWER: Acknowledge message. If recording of a live vessel's movements is necessary, then a message is sent to the MANAGE-ACTION-REQUIRED-LIST process to notify the watchsupervisor when the recording is completed.

MESSAGE TYPE: Delete-Scenario

FUNCTION: The process deletes a scenario record.

INPUT: Name of scenario record.

ANSWER: Acknowledge message.

Whenever the SET-UP-SCENARIO process is performing one of the above functions, it locks out the scenario record so that the PLAYBACK-SCENARIO process may not access the scenario record. Similarly, if the PLAYBACK-SCENARIO process is currently accessing the scenario record, it will be locked out to prevent changes by the SET-UP-SCENARIO process.

SET-UP-SCENARIO control/data paths are shown in Figure 5-20.



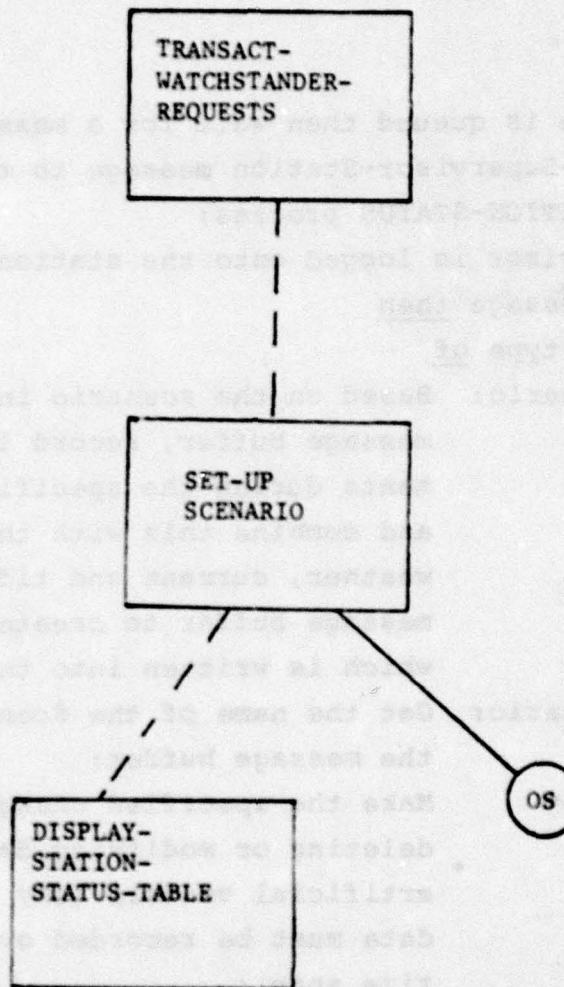


FIGURE 5-20. SET-UP SCENARIO

Process SET-UP-SCENARIO

begin

while true do

begin

if no message is queued then wait for a message;

Send a Get-Supervisor-Station message to the ACCESS-  
DISPLAY-STATION-STATUS process;

if the supervisor is logged onto the station which sent the  
Scenario-Message then

case message type of

Record-Scenario: Based on the scenario information in the  
message buffer, record live vessel move-  
ments during the specified time period,  
and combine this with the simulated vessel,  
weather, current and tide data from the  
message buffer to create a Scenario-Record  
which is written into the data base;

Modify-Scenario: Get the name of the Scenario-Record from  
the message buffer;

Make the specified changes by adding,  
deleting or modifying data on live or  
artificial vessels (any new live vessel  
data must be recorded over a specified  
time span);

Write the modified Scenario-Record back  
to the data base;

Delete-Scenario: Find the Scenario-Record with the specified  
name and remove it from the data base;

end;

Send the answer acknowledging the message;

end;

end



containing vessel position, course and speed.

- 2) Messages containing simulated sensor data are sent to the ACCESS-ENVIRONMENTAL-DATA process.

### 5.5.3 PLAYBACK-SCENARIO Process

This process is used to playback a scenario which has been stored on a scenario record. It receives messages from the TRANSACT-WATCHSTANDER-REQUESTS process at the Watch Supervisor's display station. (See Figure 5-21 for the PLAYBACK-SCENARIO control/data paths.)

#### Message Types:

MESSAGE TYPE: Initialize-Scenario

FUNCTION: The process prepares to start the scenario.

INPUT: 1) Scenario name.

2) ID of display station which will receive simulated data.

3) Flag which indicates whether to pre-load artificial vessel and passage files.

ANSWER: Acknowledge message.

MESSAGE TYPE: Start-Scenario

FUNCTION: The process starts the scenario playback.

INPUT: 1) Time relative to beginning of scenario at which to begin playback.

2) Playback speed (1X, 6X, 60X relative to the recorded speed).

ANSWER: Acknowledge message.

OTHER OUTPUT: 1) Messages are sent to MANAGE-MAP-DISPLAY process of trainee's display station,



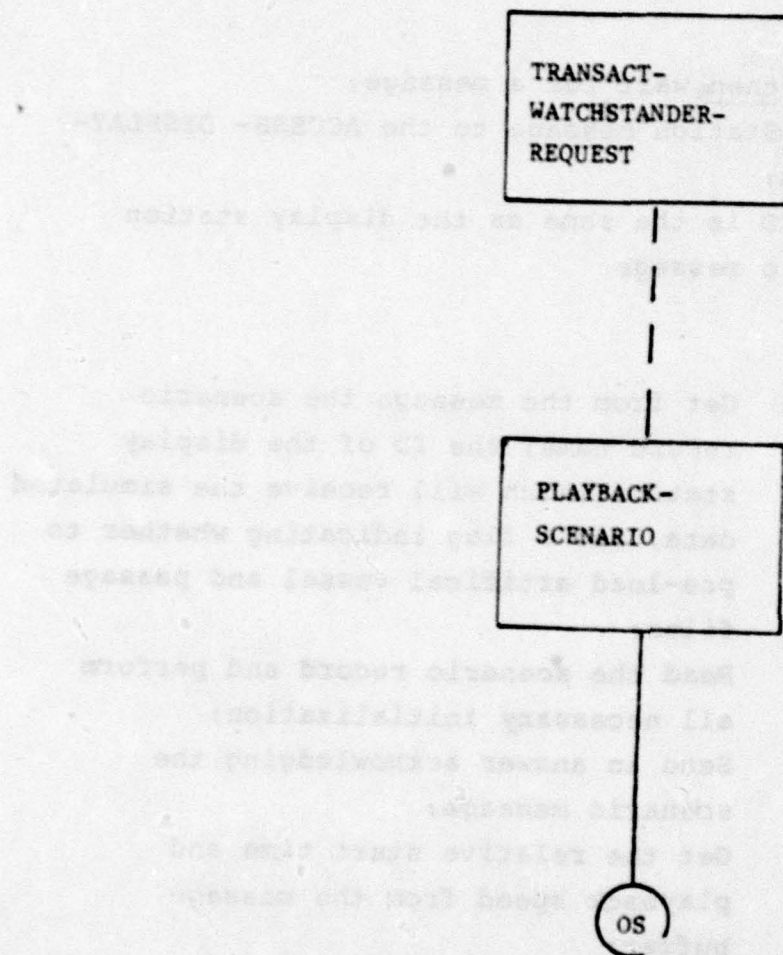


FIGURE 5-21. PLAYBACK-SCENARIO

begin

begin

```
send a Get-Supervisor-Station message to the ACCESS- DISPLAY-
STATION-STATUS process;
```

then

```
Initialize-Scenario:  Get from the message the scenario
                      record name, the ID of the display
                      station which will receive the simulated
                      data, and a flag indicating whether to
                      pre-load artifical vessel and passage
                      files;
```

**Start-Scenario:**

Send an answer acknowledging the message;

end;

**end;**

5-122



Software testing has long been accepted as a necessary part of a software system development effort. With the ultimate goal of demonstrating that the software system is error-free, it first attempts to uncover software errors.

Ideally, testing would show conclusively that the system had been specified, designed and implemented so that it met all real requirements. Moreover, we would like to be able to develop and carry out a test that would locate all errors and then, completed without error, guarantee that no errors would ever be found.

However, as Dijkstra\* has noted, "Program testing can be used to show the presence of bugs, but never to show their absence." This is true for relatively simple programs because testing all possible inputs and sequences of inputs would require more than the lifetime of the system to complete. For a complete real-time system, such as the VTS Processing/Display Subsystem, a truly exhaustive test would have to check all possible inputs, sequences of inputs, and all machine states. The number of possible combinations to be tested would be so large as to be totally incomprehensible.

\*Dijkstra, E. W., "Notes on Structured Programming", in Dahl, O. J., Dijkstra, E. W., Hoare, C. A. R., Structured Programming, Academic Press, New York, 1972.

Since testing cannot completely assure that software is correct, Dijkstra and a number of other researchers have suggested correctness proofs as a means of assuring that software will perform as it should. The concept of using proof has merit but is hardly practical or totally reliable at the current state-of-the-art\*.

Although testing is always inconclusive in showing that a software system is error-free, thorough testing is currently the only practical way of gaining confidence that our software will operate properly. Thus, testing can allow us to reduce the probability of bugs to an acceptable level.

\*Goodenough, J. B., Gerhart, S. L., "Toward a Theory of Test Data Selection", IEEE Transactions on Software Engineering, June, 1975, reprinted in Minew, E. G., Program Testing Techniques Tutorial, IEEE Catalog No. EHO 130-5, November, 1977.



## 6.1 TOP-DOWN AND BOTTOM-UP TESTING

Two general philosophies of testing, top-down and bottom-up testing, have received considerable attention. Both approaches have merit.

Top-down testing is based on the strict hierarchy approach to software design, implementation and, hence, to testing. Testing is performed on the top level of the hierarchy using stubs for lower level modules. As modules are completed, the stubs are replaced and tests repeated until all levels of the hierarchy are completed and tested. The top-down approach eliminates the need for special test drivers since modules are added after the routines which call them. Special software may have to be written to simulate the action of stubs, however, if the testing is to be meaningful.

The bottom-up approach to testing is based on the principle of building a system up from building blocks which have already been shown to be reliable. This approach is particularly useful if each lower level forms a new abstract machine which is assumed by the next higher level.

Both approaches have been used with reasonable success. There is little hard evidence on which to base a selection of one over the other. We would suggest, however, that perhaps the bottom-up approach should be applied to the testing of an operating system which involves increasingly higher levels of abstract machines. Top-down development and testing, on the other hand, appears to be more applicable to applications software that can assume a relatively high level abstract machine.

These techniques and others, such as inside-out and most critical first, are useful for testing during software development. A number of other factors should be considered in designing tests for software.

## 6.2 TESTING DURING DEVELOPMENT

Whether top-down, bottom-up or some other order of development and testing is chosen, we need testing techniques which give confidence that once a level or a module has been successfully tested, we can depend on it during later tests. If errors are encountered in later testing, we would like to be relatively certain that the bug is in those portions which have been most recently added.

To accomplish the goal of gaining confidence in modules as they are completed requires an orderly testing procedure that can uncover the majority of the bugs which may exist. One approach to testing would treat a module or a group of modules as a black box to which we would apply a randomly selected set of inputs. We would then observe the outputs to determine if they represented the correct outputs.

A black box approach, however, cannot provide a high degree of confidence. Without knowledge of the internal structure of a black box, we cannot infer that a particular input will yield a correct output because some other input produced a correct output. Without being able to determine that all values in a given range are handled identically by the black box, we would have to test all possible combinations of input\*. For all non-trivial cases, this is impossible.

---

\*Note that by input we actually mean everything that may influence the result. This includes the inputs and the state of the machine such as internally stored data.



Another approach, which is practical, considers the structure of the software in selecting test data. The central notion is to assure that the tests will cause all portions of the software to be executed at least once. This is not sufficient to assure that the software is error free, but certainly we will have more confidence in a portion of our software if it has executed correctly for at least one case than if it has never been executed at all.

A related technique attempts to test all the boundary conditions that are relevant to the module or modules under test. For example, if the module should reject input values outside the range 0 to 100, we would test values on either side of 0 and on either side of 100. Similarly, we would select values which would include all of the cases which we would expect to be handled differently by the program under test.

At the lower levels, these two techniques are equivalent. If we identify all the boundary conditions, we will force the program to traverse all paths through the program. The first technique is perhaps better and more thorough for testing a particular program. The second technique, however, can be used to develop test data from a detailed specification without direct knowledge of the program itself. In that sense, one technique is perhaps better for debugging while the other may be better for functional testing.

### 6.3 FUNCTIONAL TESTING

When a system or a significant subsystem has been completed, functional testing is needed to assure that the system or subsystem meets the detailed functional requirements.

For VTS, it will be necessary to develop a set of functional tests based on the state diagrams presented in Part I of this document. The tests should demonstrate all of the states and all of the transitions shown in each state diagram. As an example, tests of the state diagram shown in Figure 6-2 of Part I follow.

The inactive state (State 1 in Figure 6-2) is the initial state of the display station. Our tests would include:

- . Enter all valid functions, other than enter passwords. Verify that the functions are not accepted.
- . Enter an invalid password. Verify that the invalid password is rejected.
- . Enter a valid password. Verify that the password is accepted and a ready indicator displayed.
- . From the ready state, enter ~~ON~~ OFF command and repeat the above tests to verify that the station has returned to the inactive state.

Similar tests would be specified for every state and transition shown. The final test specification would, of course, provide detailed instructions for such tests which would require detailed knowledge of the display station hardware to develop.



Functional tests of the operator invoked functions can be performed by entering function codes and data in the normal manner. A completely controlled and repeatable test can be run on these functions using only those facilities which are required for the operational system.

For other functions such as radar input processing, the task of developing a functional test must include consideration of providing testing tools. To provide a controlled and repeatable test of radar input processing, it is not sufficient to connect an operational radar and process its inputs. Such a test would not be repeatable and some unusual conditions might never be tested.

Recorded radar data can be used to provide a controlled and repeatable test. Such testing is highly desirable. Additional testing, using simulated data will almost certainly be required to validate functions such as merge processing.

#### 6.4 LOAD TESTING

For programs that do not operate in real time, thorough, controlled and repeatable functional tests are sufficient to provide confidence that the software will operate properly. For real-time software, we must also demonstrate that the software will perform properly under load.

Load testing should be designed to show that:

- . Response time requirements are met under the maximum design load.
- . The system does not "blow" or deadlock at any conceivable load.
- . Facilities provided for load shedding or prioritizing system functions perform as required.

Load tests, to meet these goals, require a facility for creating a controllable, sustained load. For VTS, the number of vessels in the coverage area is the principle source of processing load. Such a load can perhaps be created most effectively by simulating radar input data. The internal simulation facility can be useful, but is not completely satisfactory since such data is known to be artificial by the system and is treated somewhat differently by the system.

Two alternatives could be considered for generating a realistic load of vessel traffic. The normal simulation facility could be modified so that for testing purposes, the data could be treated normally and a more extensive load simulated. Alternatively, the load could be generated by an external simulator.



To the maximum practical extent, the artificial load should represent a realistic mix of events. For example, the percentage of vessel pairs which would be in a collision alert should be realistic. It is also essential to be able to create a load that can force the system to its limits in a controlled fashion.

Simulated inputs can also be used to test the ability to handle watchstander input loads. The VTS architecture will allow one display station processor or a spare processor to be used to emulate multiple display stations.

The VTS Processing/Display Subsystem is amenable to extensive automation of testing. However, the degree of automation that will be cost effective is difficult to ascertain at this point since it is not clear how rapidly VTS systems will be proliferated and it is also not clear how frequently software revisions would require extensive retests of the software.

An automated testing facility could be built for VTS using a redundant processor which is attached to the bus. The processor could simulate a substantial number of display stations by sending data to the appropriate processors and checking data returned. The same processor could also be connected to radar and other sensor input ports so that it could simulate realistic inputs from those sensors. Extensive scenarios could be developed that would effectively perform both functional and load tests.

Nevertheless, the cost of extensive automation of testing could be substantial. Creating the kind of complex scenarios needed would require a highly complex software package which would itself be difficult to test. We suggest, therefore, that the initial development include a basic load generation facility. The development of more elaborate testing tools could then be deferred until the costs can be justified.

## 6.5 PERFORMANCE TESTING AND EVALUATION

Performance testing will also be important for the VTS Processing/Display Subsystem. To meet the requirements of a particular VTS, it is sufficient to demonstrate that the design loads can be handled and that response time requirements can be met at the design load.

Determining system limits and measuring the actual resource utilization resulting from a variety of load factors will also be important. Data gathered from such tests will be essential in determining the actual hardware requirements for additional VTS configurations.

The analysis of various VTS configurations has been based on estimates of resource requirements and the assumption of a hypothetical set of hardware\*. Such estimates are a rough approximation at best. Performance testing allows measurements to be taken that will provide a more realistic basis for future analysis.

\*Henson, C. C., Mickey, F. T., Graham, R. S., McIntosh, B. A.,  
VTS Processing/Display Subsystem Design, January, 1979.



## 6.6 RECONFIGURATION TESTING

The VTS Processing/Display Subsystem will include the software for extensive error and failure detection and software to effect reconfiguration. Thorough testing of these features is extremely important, but extremely difficult as well.

Some failures such as the complete failure of a processor or a bus can be "caused" easily and the system response observed. Other failures such as main memory faults are usually difficult to actually induce without physically damaging the hardware. It may be impractical, therefore, to test the recognition of such errors in a realistic way.

For those faults which can be induced, our testing will include these failures. For faults which cannot actually be induced safely, tests must be devised which bypass the actual fault recognition mechanism and allow the fault to be simulated.

Initial tests of reconfiguration and recovery can be done by powering down a processor while it is performing a major system function. We can also shut off a disc drive or unplug one of the buses. Nearly all actual devices can be physically disconnected to simulate a total failure of the device. System reconfiguration and recovery strategy can then be tested for complete failure of any single device.

Additional tests will then be needed to show that the system can properly recognize less drastic failures and that it can correlate a variety of symptoms to determine which device has failed.

## A.1 RELATIVE POSITION EQUATIONS

The purpose of these equations is to determine the distance and bearing from point 1 to point 2, given the coordinates of each point.

The variables are defined as follows:

- $L_1$  = latitude at point 1
- $L_2$  = latitude at point 2
- $l_1$  = longitude at point 1
- $l_2$  = longitude at point 2
- $d$  = distance in miles of the great circle between point 1 and point 2
- $c$  = the compass bearing in degrees at point 1 toward point 2 along the great circle.

If point 1 and point 2 are a great distance apart, spherical trigonometry is used to take into account the earth's curvature. The spherical trigonometric equations for distance and bearing are:

$$d = 138.342 \operatorname{Arcsin} \sqrt{\sin^2 \frac{l_2 - l_1}{2} \cos L_1 \cos L_2 + \sin^2 \frac{L_1 - L_2}{2}}$$

$$c = 2 \operatorname{Arcsin} \sec L_1 \csc \frac{d}{69.171} \sin^2 \frac{90^\circ - L_2}{2} - \sin^2 \frac{|D + L_1 - 90^\circ|}{2}$$



The threshold distance beyond which the spherical equations should be used is the distance at which the permissible error is equal to the maximum difference in results between the plane geometry equations and the spherical geometry equations.

## A.2 CPA EQUATIONS

The purpose of these equations is to determine the distance, time, and positions at CPA of two objects, where at least one of the objects has a non-zero velocity.

Definitions to be used in the equations are:

- $X_1$  = longitude of object 1
- $X_2$  = longitude of object 2
- $Y_1$  = latitude of object 1
- $Y_2$  = latitude of object 2
- $V_{x1}$  = longitudinal velocity component of object 1
- $V_{x2}$  = longitudinal velocity component of object 2
- $V_{y1}$  = latitudinal velocity component of object 1
- $V_{y2}$  = latitudinal velocity component of object 2
- $\bar{I}$  = unit longitude vector
- $\bar{J}$  = unit latitude vector

The dot product of two vectors A and B is  $\bar{A} \cdot \bar{B} = |\bar{A}| |\bar{B}| \cos \theta$ , where  $\theta$  = angle between vectors  $\bar{A}$  and  $\bar{B}$ .

The relative position vector is defined as:

$$\bar{D} = (X_2 - X_1) \bar{I} + (Y_2 - Y_1) \bar{J}$$

The relative velocity vector is defined as:

$$\bar{V} = (V_{x2} - V_{x1}) \bar{I} + (V_{y2} - V_{y1}) \bar{J}$$



The dot product  $\bar{V} \cdot \bar{D}$  is calculated. If the result is positive, the CPA point has already been passed (the two objects are now diverging, rather than converging); otherwise, the time to CPA is computed with the following formula.

$$T_{cpa} = \frac{-\bar{V} \cdot \bar{D}}{\bar{V} \cdot \bar{V}}$$

If  $T_{cpa}$  is very great the objects are traveling nearly at the same velocity and distance. In this case, the following calculations will not be very accurate.

The miss distance, or range at CPA is calculated as follows:

$$R_{cpa} = (\bar{D} + \bar{V} T_{cpa}) \cdot (\bar{D} + \bar{V} T_{cpa})^{\frac{1}{2}}$$

The positions of the 2 objects at CPA are computed with the following equations:

$$\bar{P}_{1cpa} = \bar{D}_1 + \bar{V}_1 T_{cpa} \quad \text{for object 1}$$

$$\bar{P}_{2cpa} = \bar{D}_2 + \bar{V}_2 T_{cpa} \quad \text{for object 2}$$

### A.3 DEAD-RECKONING COMPUTATIONS

There are two situations considered in this section: 1) dead-reckoning a vessel not following a route, and 2) dead-reckoning a vessel following a route consisting of a series of connected straight line segments.

The computation of dead-reckoned position for the first case is trivial. It is computed with the equation:

$$\bar{P} = \bar{D} + \bar{V}t$$

where

$\bar{P}$  = the dead-reckoned position

$\bar{D}$  = the present position

$\bar{V}$  = the present velocity

$t$  = the time interval

For a vessel following a route, the following procedure is used to compute the position along the route.



Procedure DEAD-RECKON-ALONG-ROUTE

begin

t: = time interval for dead-reckoning;

$\bar{D}$ : = present vessel position;

$\bar{V}$ : = present vessel velocity;

i: = index of next straight line endpoint in route;

Quit: = false;

while Quit = false do

begin

$\bar{D}_i$  = position of endpoint;

d =  $\bar{V} t$  ; /\*distance vessel will travel\*/

$d_i$  =  $\bar{D}_i - \bar{D}$  ; /\*distance to next endpoint\*/

if d  $\leq$   $d_i$  ; /\*if vessel will pass endpoint before  
dead-reckon point\*/

then

d: = d -  $d_i$  ; /\*compute distance remaining past this  
endpoint\*/

i: = i+1 ; /\*increment index to endpoint of  
following straight line segment\*/

else

$\bar{P}$ : =  $\frac{(\bar{D}_i - \bar{D}_{i-1})}{\bar{D}_i - \bar{D}_{i-1}} d$  ; /\*compute vessel position along this  
segment\*/

Quit: = true;

end;

end

#### REFERENCES

- 1) Bowditch, Nathaniel, American Practical Navigator, U.S. Naval Oceanographic Office, Washington, D.C., 1966, pp.232-234
- 2) Beyer, William, ED., C.R.C. Standard Mathematics Tables, CRC Press, Inc., West Palm Beach, Florida, 1978.
- 3) VTS Operation and Maintenance Manual, Vol.V, Part 2, December 1973, pp.2-583 and 2-584.